Payment Solutions: XIP QuadSPI Lab Guide

Rev. 0.2

SECURE CONNECTIONS FOR A SMARTER WORLD

1 Purpose

This document is provided as a hands-on lab guide for the NXP Technology day in Irvine California. The intent of the lab is to demonstrate how the SLN-POS-RDR uses external Serial Flash for high performance memory expansion. For the case of this lab, we will use the FRDM-K82F as the target hardware.

The lab will direct the user through updating IAR project debug settings to support single button download and debug to external serial flash. Then, it will run a couple of benchmark scenarios that show how proper code placement and CPU architecture allows for high performance execute in place from external memory.

By the end of this lab the user will learn how to use:

- IAR Flash loader projects to support external serial flash
- Linker file configurations for code and data placement
- Mbed TLS cryptography benchmark application

2 Resources

The following resources are for reference. If you could review the below before completing the lab it is highly recommended.

https://community.nxp.com/community/training/ftf-2015-training-presentations/projects/leveraging-the-scalability-of-kinetis-mcu-memory-resources

3 Overview and Essential Background

Bench Setup

This LAB uses the FRDM-K82F and a PC running Windows. The FRDM-K82F has been updated to use the Segger JLink profile for the on-board debugger CMSIS-DAP.

This LAB assumes that the PC is preloaded with the IAR Embedded work bench version 8 and the related drivers. The path to the Lab Software is:

C:\NXP\ SDK_2.2_FRDM-K82F_Irvine_Training

In this lab, only a USB connection to the debug port is needed.

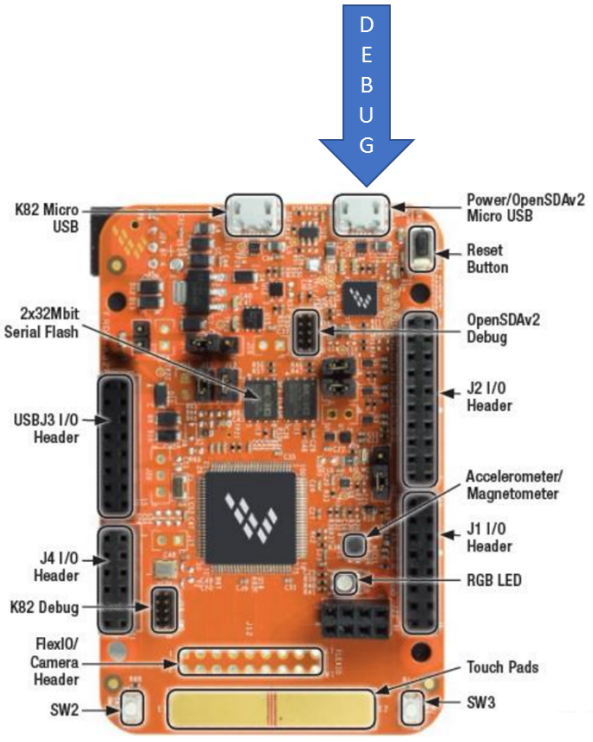


Figure 2. FRDM-K82F primary component placement

Background on Memory Expansion

Kinetis MCUs address the challenge of supporting memory expansion with

- Intelligent use of Cortex-M4 core micro-architecture and its predefined system memory map
- L1 Code and System Caches for mitigating slower memory interfaces
- Highly capable memory controllers with built-in buffers

The following diagrams show how the system memory map for the Kinetis MCU uses the CPU buses to support high performance and debugging for external memory.

Table 1. K80/ K81System memory map

System 32-bit Address Range	Destination Slave	Access
0x0000_0000-0x03FF_FFFF [1]	Program flash and read-only data (Includes exception vectors in first 1024 bytes)	All masters
0x0400_0000-0x07FF_FFFF	QSPI0 (Aliased area) Mapped to the same access space of 0x6800_0000 to 0x6FFF_FFFF	Cortex-M4 core (M0) only
0x0800_0000-0x0FFF_FFFF	SDRAM (Aliased area) Mapped to the same access space of 0x8800_0000-0x8FFF_FFFF To alias this space to 0x8800_0000 - 0x8FFF_FFFF , set the appropriate SDRAMC chip select's address mask bit31.	Cortex-M4 core (M0) only
0x1000_0000-0x17FF_FFFF	Reserved	-
0x1800_0000-0x1BFF_FFFF	FlexBus (Aliased Area) Mapped to the same access space of 0x9800_0000-0x9BFF_FFFF To alias this space to 0x9800_0000 - 0x9BFF_FFFF, set the appropriate FlexBus chip select's address mask bit31.	Cortex-M4 core (M0) only
0x1C00_0000–0x1C00_7FFF	ROM	All masters
0x1C00_8000–0x1FFF_FFFF[2]	Tightly Coupled Memory Lower (TCML) SRAM_L: Lower SRAM (ICODE/ DCODE)	All masters
0x2000_0000–0x200F_FFFF[3]	Tightly Coupled Memory Upper (TCMU) SRAM_U: Upper SRAM bitband region	All masters
0x2100_0000-0x21FF_FFFF	Reserved	–
0x2200_0000-0x23FF_FFFF	Aliased to TCMU SRAM bitband	Cortex-M4 core only
0x2400_0000-0x2FFF_FFFF	Reserved	–
0x3000_0000-0x33FF_FFFF	Flash Data Alias	Cortex-M4 core only
0x3400_0000-0x3FFF_FFFF	Reserved (was Flash Flex Alias on K60_2MB)	Cortex-M4 core only
0x4000_0000-0x4007_FFFF	Bitband region for AIPS0	Cortex-M4 core & DMA
0x4008_0000-0x400F_EFFF	Bitband region for AIPS1	Cortex-M4 core & DMA
0x400F_F000-0x400F_FFFF	Bitband region for GPIO	Cortex-M4 core & DMA

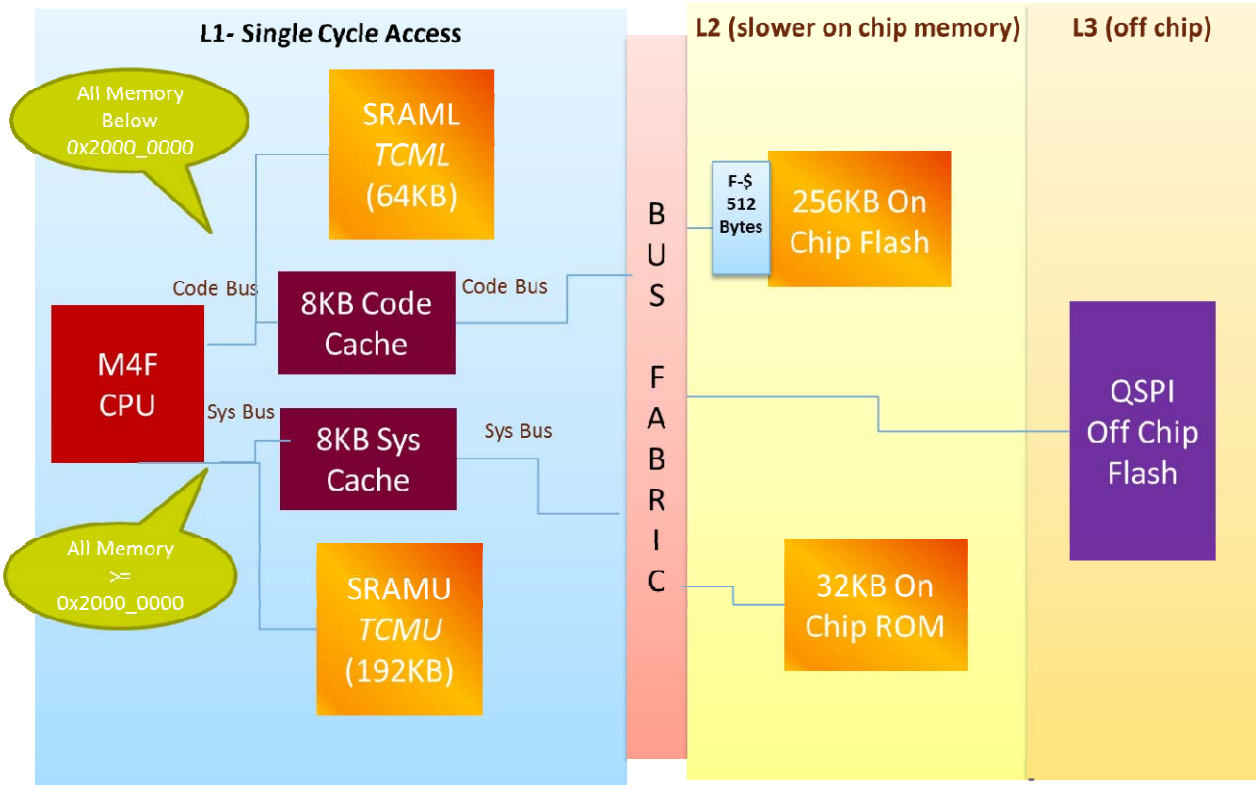
QuadSPI memory region below 0x2000_0000 should be used for Code

Table 1. K80/ K81System memory map (continued)

System 32-bit Address Range	Destination Slave	Access
0x4010_0000-0x41FF_FFFF	Reserved	–
0x4200_0000-0x43FF_FFFF	Aliased to AIPS and GPIO bitband	Cortex-M4 core only
0x4400_0000-0x5FFF_FFFF	Reserved	–
0x6000_0000-0x66FF_FFFF	FlexBus (External Memory - Write-back)	All masters
0x6700_0000-0x677F_FFFF	QuadSPI0 Rx Buffer	All masters
0x6780_0000-0x67FF_FFFF	Reserved	All masters
0x6800_0000-0x6FFF_FFFF	QSPI0 (External Memory)	All masters
0x7000_0000-0x7FFF_FFFF	SDRAM (External Memory - Write-back)	All masters
0x8000_0000-0x87FF_FFFF	SDRAM (External Memory - Write-through)	All masters
0x8800_0000-0x8FFF_FFFF	SDRAM (External Memory - Write-through)	All masters
0x9000_0000-0x97FF_FFFF	Reserved	-
0x9800_0000-0x9FFF_FFFF	FlexBus (External Memory - Write-through)	All masters
0xA000_0000-0xDFFF_FFFF	FlexBus (External Peripheral - Not executable)	All masters
0xE000_0000-0xE00F_FFFF	Private Peripherals	Cortex-M4 core only
0xE010_0000-0xFFFF_FFFF	Reserved	–

QuadSPI memory region(s) above 0x2000_0000 should be used for Data and Constants

In addition to the system memory map, the Kinetis MCU uses multiple levels of memory buffering to enhance execute in place performance. The memory architecture is represented below.



Lab sections

The major sections of this lab are:

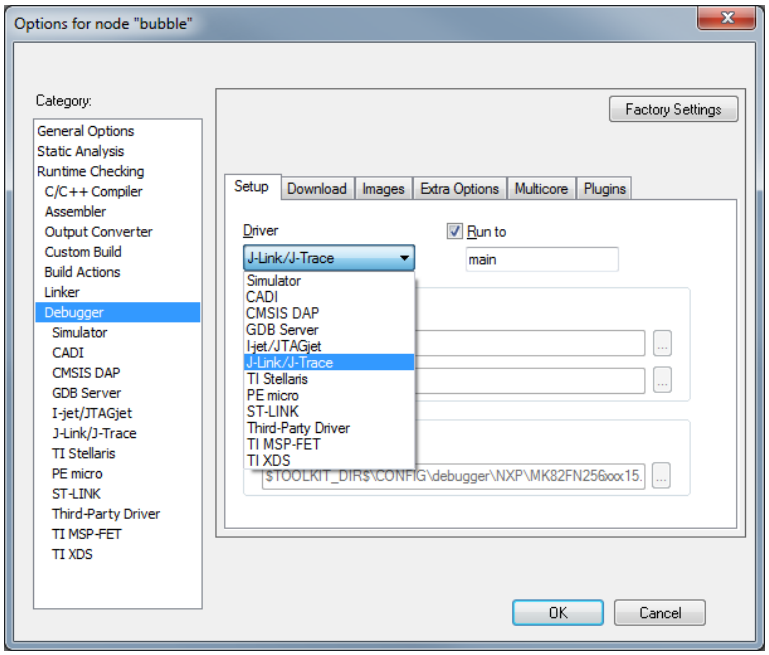
- *Running mbed TLS SHA-512 benchmark in internal Flash*
- *Updating debugger settings to support serial Flash*
- *Updating Linker file for code placement*
- *Debugging with external serial flash*
- *Enabling CPU Cache for best performance*

4 Running mbed TLS SHA-512 benchmark in internal Flash

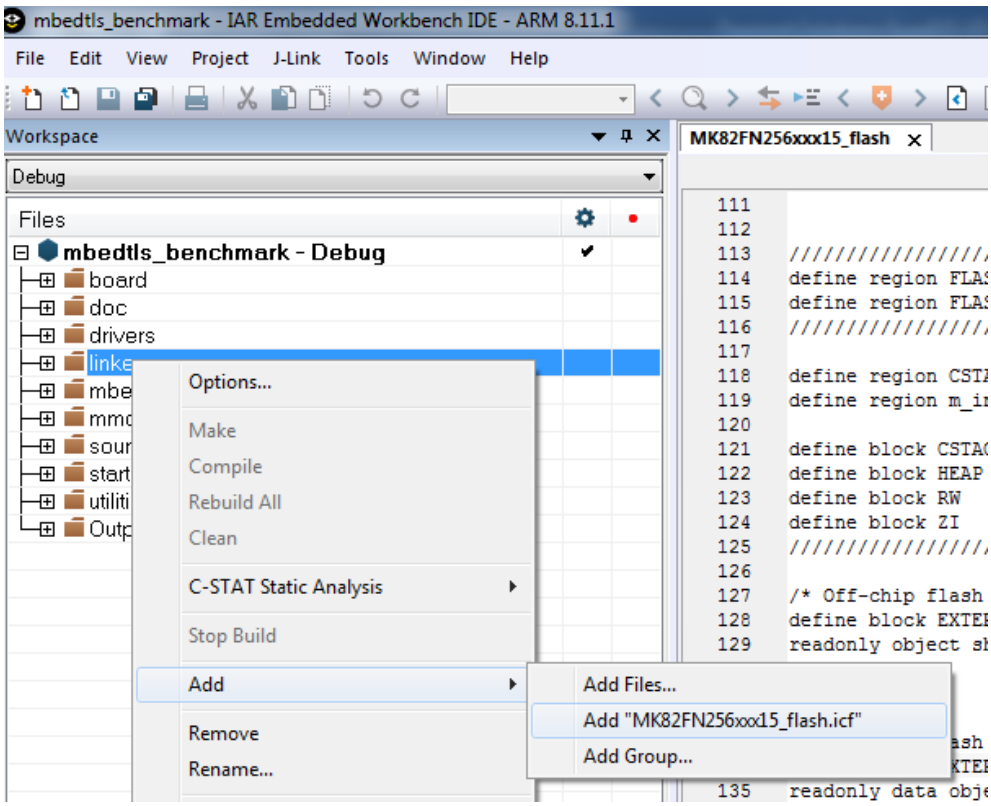
The end result of this section is that the debug terminal will show the throughput for mbed TLS SHA-512 benchmark.

The following steps **were done previously** to the default SDK software benchmark example. No Action is required for the following 2 points.

1) Change debugger to Segger JLink

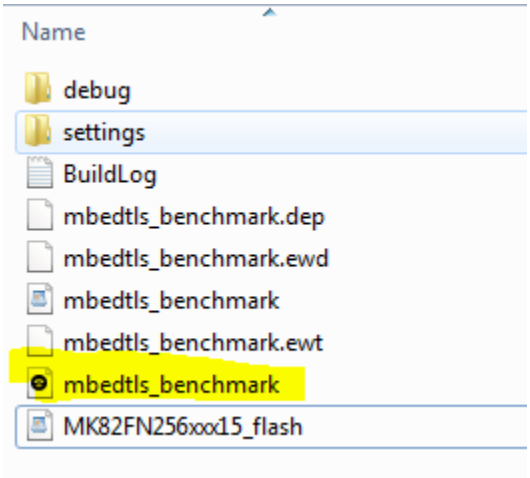


2) Add a folder with the project linker file



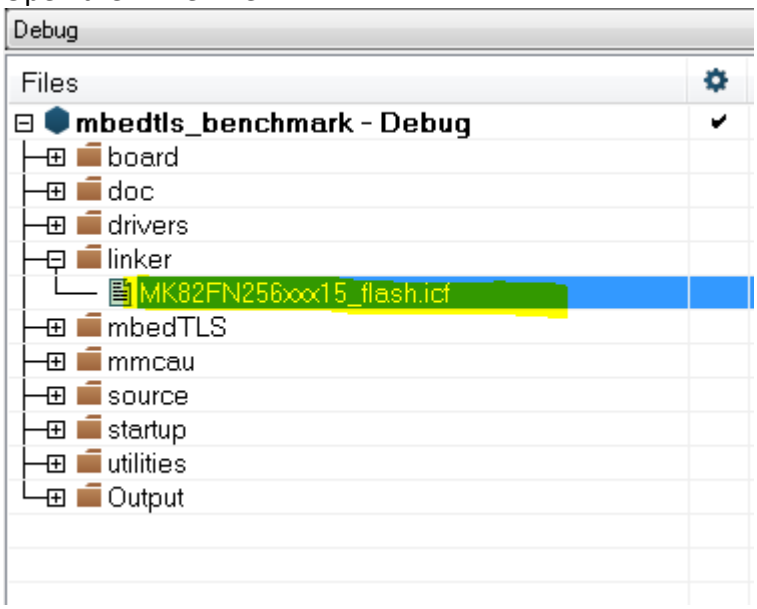
Open the application workspace

Navigate and open the SDK example by double left clicking the project at:
C:\NXP\SDK_2.2_FRDM-K82F_Irvine_Training\boards\frdmk82f\demo_apps\mbedtls\mbedtls_benchmark\iar



Examine Linker file

Open the Linker file



The following changes have been made to the linker file to prepare it for this lab.
The FRDM-K82F contains 2 Serial Flash ICs.
There is a symbol defined for Flash 1, Code Space, Flash 1 Data Space, and Flash 2 Data space. Note the addresses for these spaces.

Regions are setup for only using a single serial flash (Flash1)

```
define symbol m_text_end                = 0x0003FFFF;
//////////////////////////////////////////////////////////////////////////////
//  Application CODE in External Flash
define symbol m_flash_bank1_text_ext_start          = 0x04000000; // 2MB
define symbol m_flash_bank1_text_ext_end            = 0x041FFFFF;

// Application Data in External Flash
define symbol m_flash_bank1_data_LIB_ext_start      = 0x68200000; // 2MB
define symbol m_flash_bank1_data_LIB_ext_end        = 0x683FFFFF;

// DATA Libraries in External Flash
define symbol m_flash_bank2_data_LIB_ext_start      = 0x68400000; // 4MB
define symbol m_flash_bank2_data_LIB_ext_end        = 0x687FFFFF;
//////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////
define region FLASH_TEXT_LIB_region = mem:[from m_flash_bank1_text_ext_start to m_flash_bank1_text_ext_end];
define region FLASH_DATA_LIB_region = mem:[from m_flash_bank1_data_LIB_ext_start to m_flash_bank1_data_LIB_ext
//////////////////////////////////////////////////////////////
```

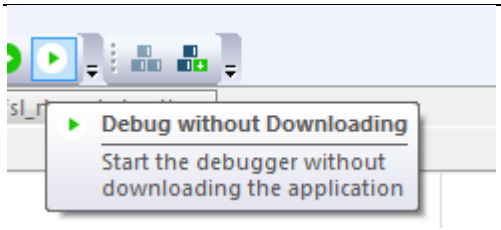
```
////////////////////
place in FLASH_TEXT_LIB_region              { block EXTERNAL_TEXT };
place in FLASH_DATA_LIB_region              { block EXTERNAL_DATA };
```

Placement of code is handled with linker file commands. For this first section, SHA512 code and data is in internal flash, so line these should be commented.

```
//////////////////////////////////////////////////////////////////////////////

/* Off-chip flash TEXT block: */
define block EXTERNAL_TEXT with fixed order {
//readonly object sha512.o

 };

/* Off-chip flash DATA block: */
define block EXTERNAL_DATA with fixed order {
//readonly data object sha512.o
//readonly object sha512.o
};

//////////////////////////////////////////////////////////////
```

Download and debug the project

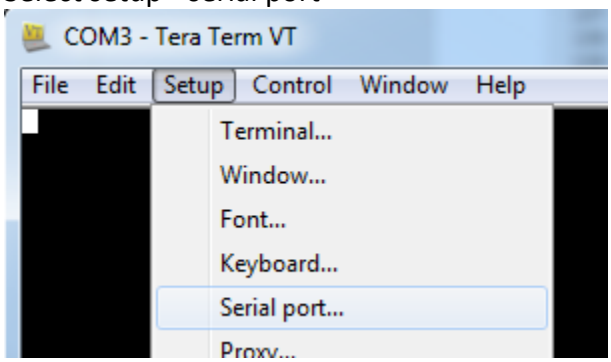
From the bootloader workspace, press the debug without downloading button



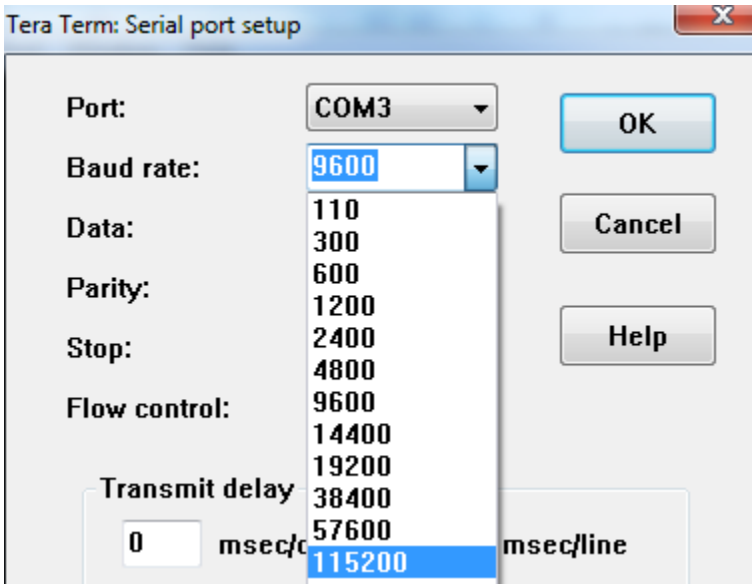
Open Tera Term (It should be on your menu bar)

Select serial and press OK

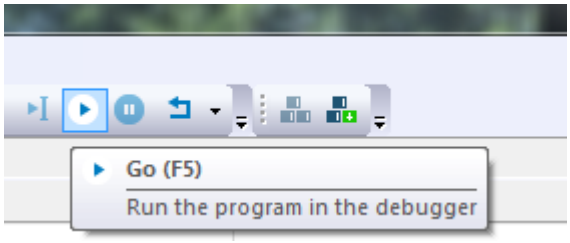
Select setup->serial port



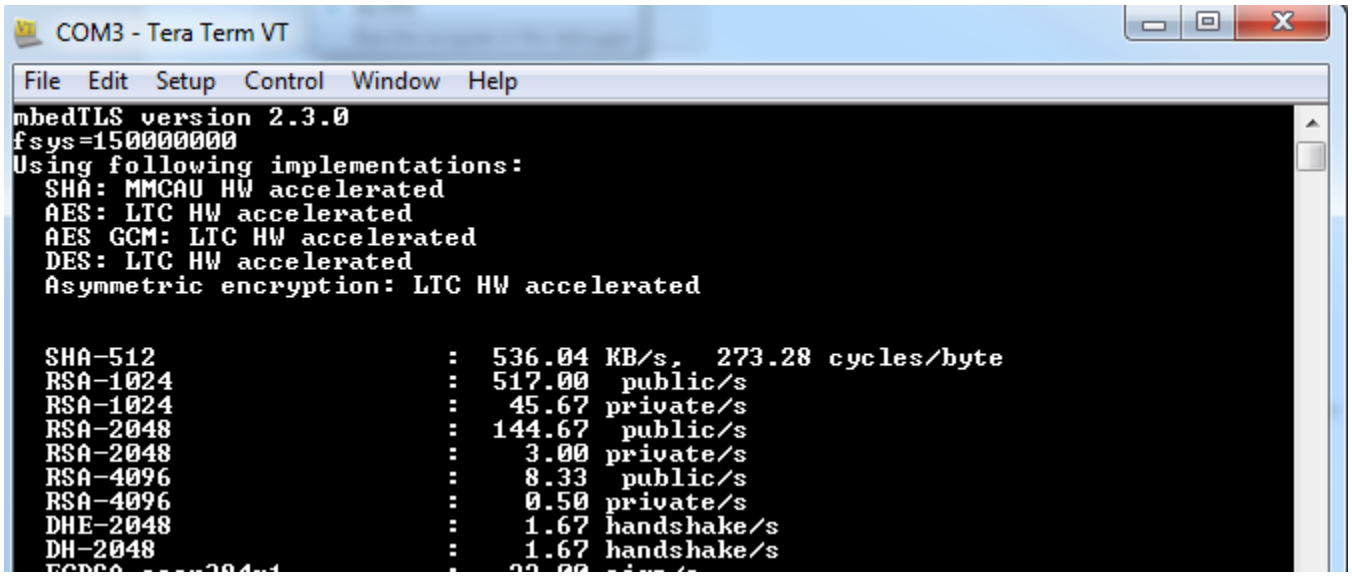
Set 115200 baud rate



From the IAR IDE Press the Go button to see normal operation.



The output should match the below



You are now ready to move on to the next section.

5 Updating debugger settings to support serial Flash

In this section, we will change the IAR debug settings to allow single button download and debug for external flash.

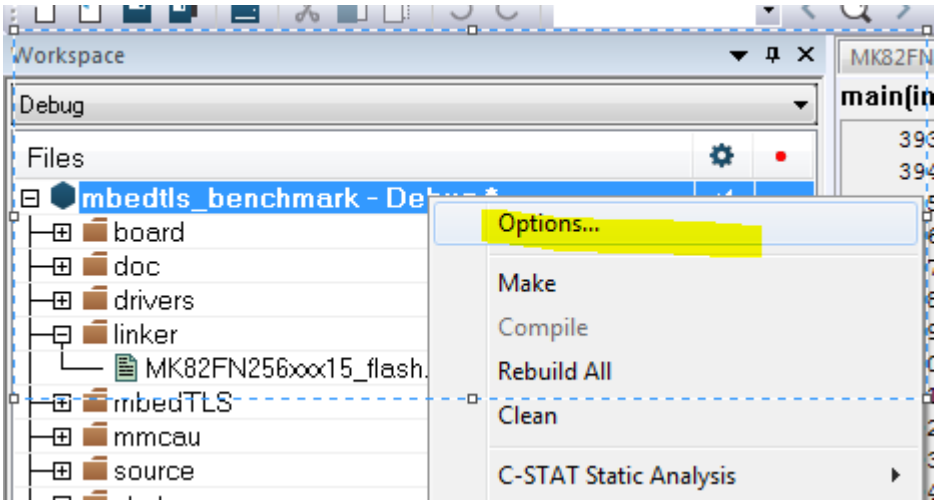
How this works depends on 4 files which define the debug and flash loading process.

These files are:
.BOARD file: This defines the memory regions and which.Flash files are assigned to each region.
.FLASH file: This defines which executable (.out) and Macro files work for the flash region
*.out file: This file is generated from an IAR project which is used to work with Segger JLINK debug tools
*.MAC file: this file works with Segger JLINK tools to do low level initialization over the debug interface

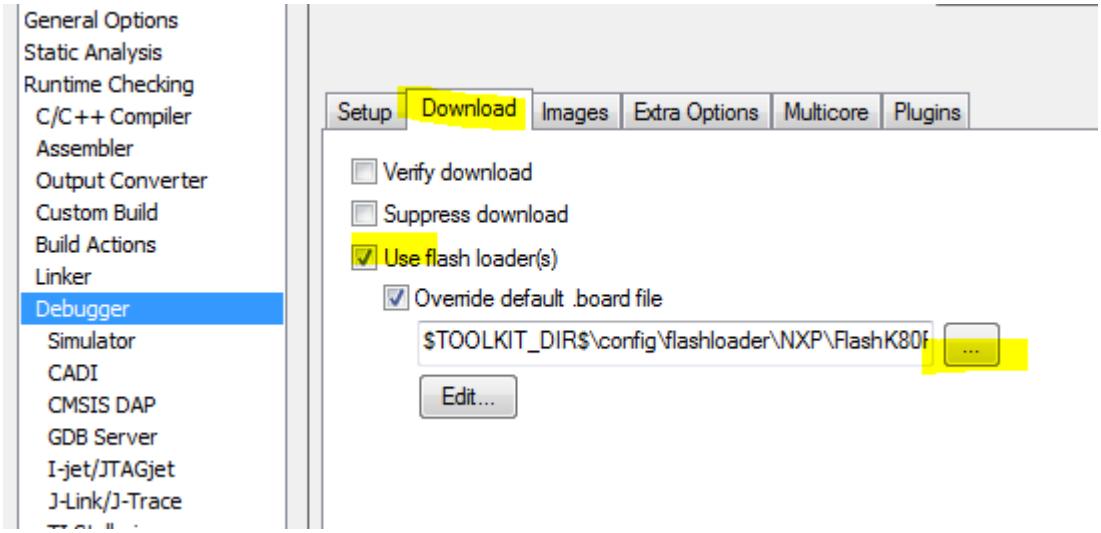
All of these files work together to allow the single button download and debug.

Update the *.Board file

Open the project options

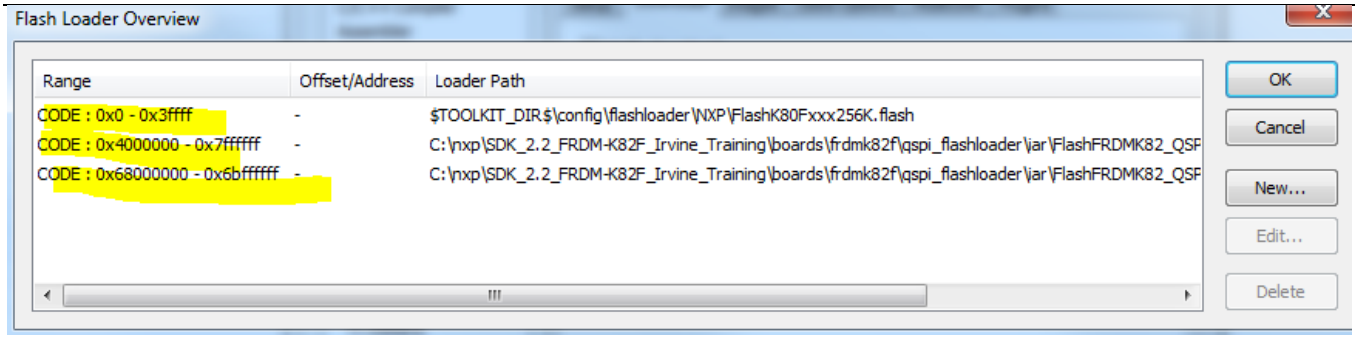


From the debugger settings, select download and click Use Flash Loader(s). Then click on the file selection button (…) to update the board file



Navigate to C:\nxp\SDK_2.2_FRDM-K82F_Irvine_Training\boards\frdmk82f\qspi_flashloader\iar and select the FlashFRDMK82F_QSPI.board file

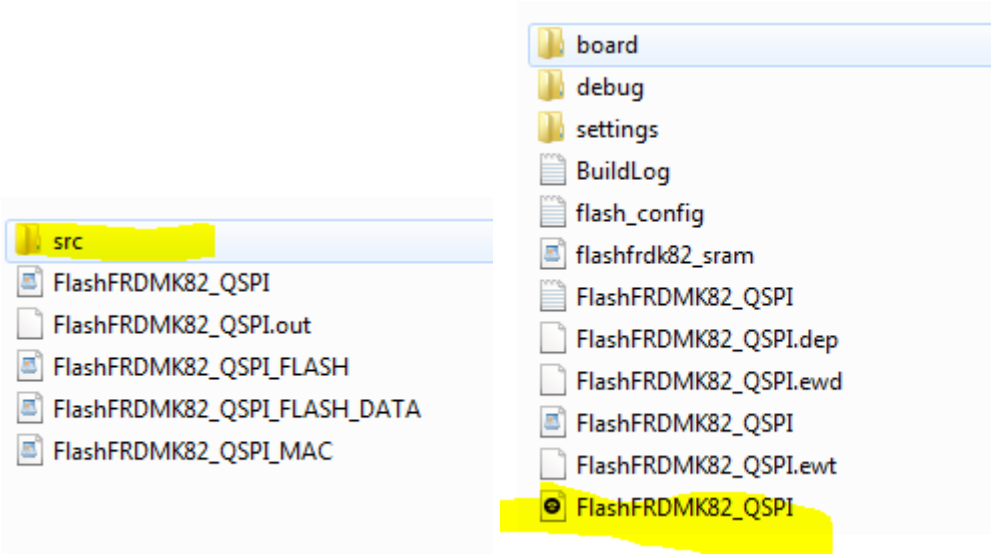
Press Edit to see the board file configuration.
Three memory regions are set and different *.Flash Files are chosen for each region.



PRESS OK on all configuration settings.

To see the *.Flash files – navigate to the below path in windows explorer.
Navigate to C:\nxp\SDK_2.2_FRDM-K82F_Irvine_Training\boards\frdmk82f\qspi_flashloader\iar

Here you can also find the IAR project for the Flashloader. This project would have to change for different serial flash devices.



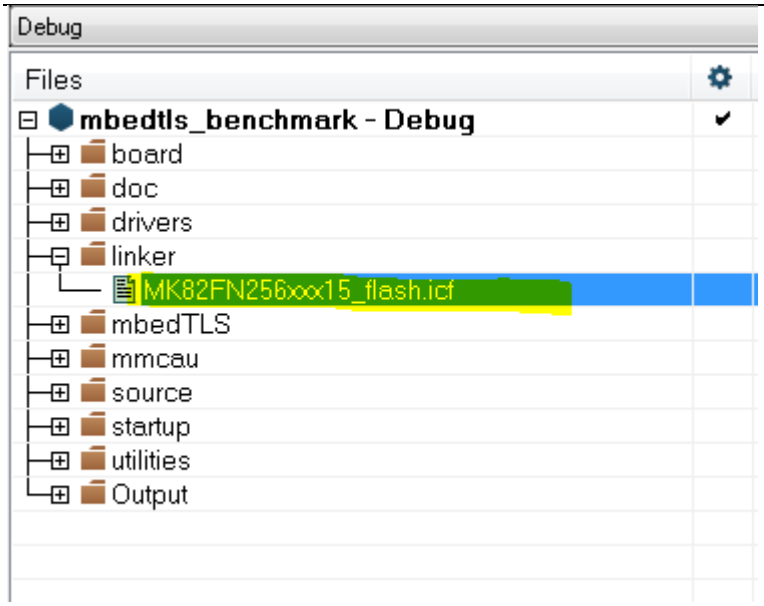
You are now ready to move on to the next section.

6 Updating Linker file for code placement

Now we will place SHA512 into external serial flash – in the Data Space of the memory map.

Open the Linker file and make changes

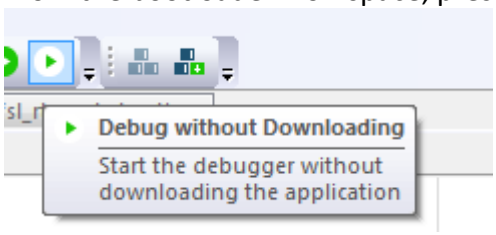
Open the Linker file



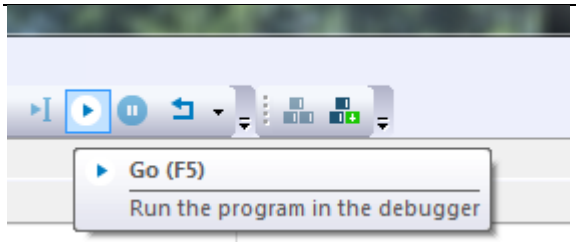
Un-Comment line 136 from the linker file.

```
125 ////////////////////////////////////////////////////////////////////////////////
126
127 /* Off-chip flash TEXT block: */
128 define block EXTERNAL_TEXT with fixed order {
129 //readonly object sha512.o
130
131  };
132
133 /* Off-chip flash DATA block: */
134 define block EXTERNAL_DATA with fixed order {
135 //readonly data object sha512.o
136 //readonly object sha512.o
137 };
138
139 ////////////////////////////////////////////////////////////////////////////////
```

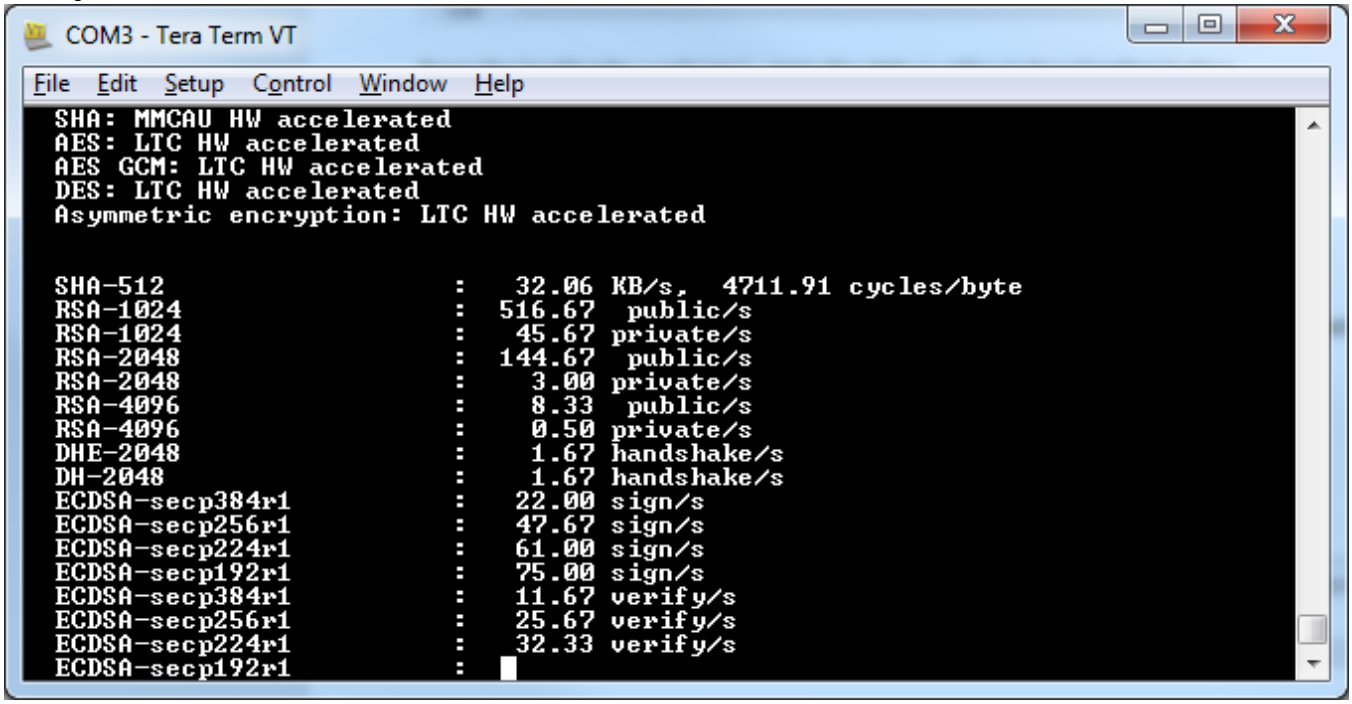
From the bootloader workspace, press the debug without downloading button



From the IAR IDE Press the Go button to see operation with SHA-512 code in external flash (Data space)



The performance is degraded. It takes >30seconds to update the output.

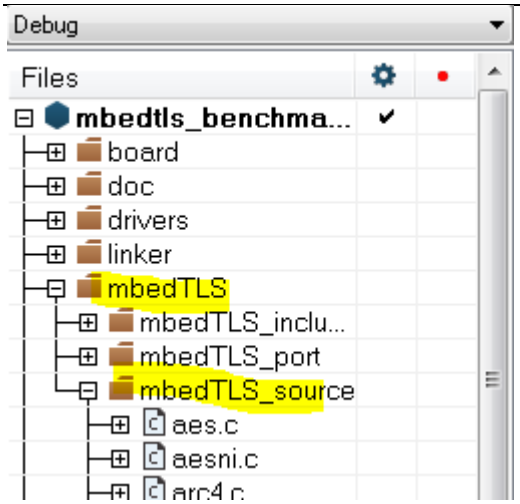


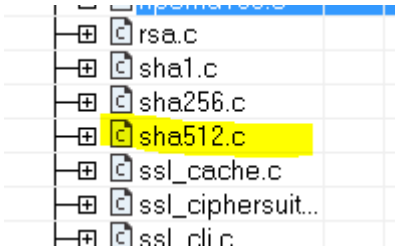
You are now ready to move on to the next section.

7 Debugging with external serial flash

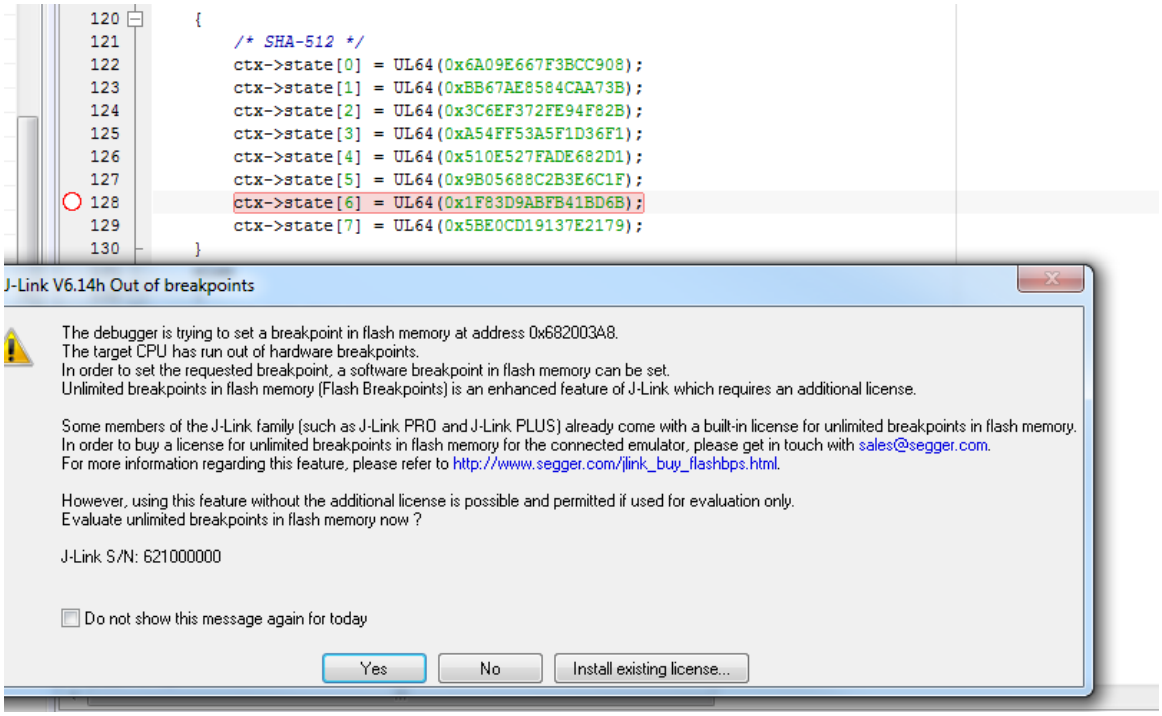
Set a breakpoint in SHA512.c

Expand the mbedTLS folder and select to open SHA512.c file





Attempt to set a breakpint at line 128 by double clicking to the left of the line number
Only SW break points can be set – **this is because the CPU does not support breakpoints in memory space**

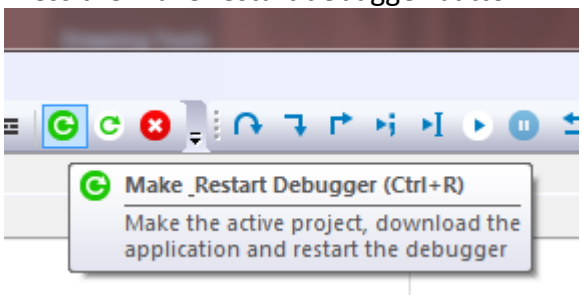


Update Linker file to relocate SHA512 code to memory space

Open the linker file.
Un-comment line 129 and 135.
Comment line 136.

```
125  ////////////////////////////////////////////////////////////////////////////////
126
127  /* Off-chip flash TEXT block: */
128  define block EXTERNAL_TEXT with fixed order {
129  readonly object sha512.o
130
131   };
132
133  /* Off-chip flash DATA block: */
134  define block EXTERNAL_DATA with fixed order {
135  readonly data object sha512.o
136  //readonly object sha512.o
137  };
138
139  ////////////////////////////////////////////////////////////////////////////////
```

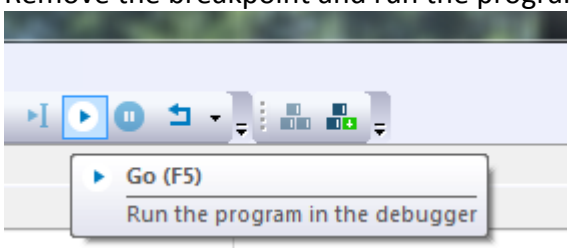
Press the make restart debugger button

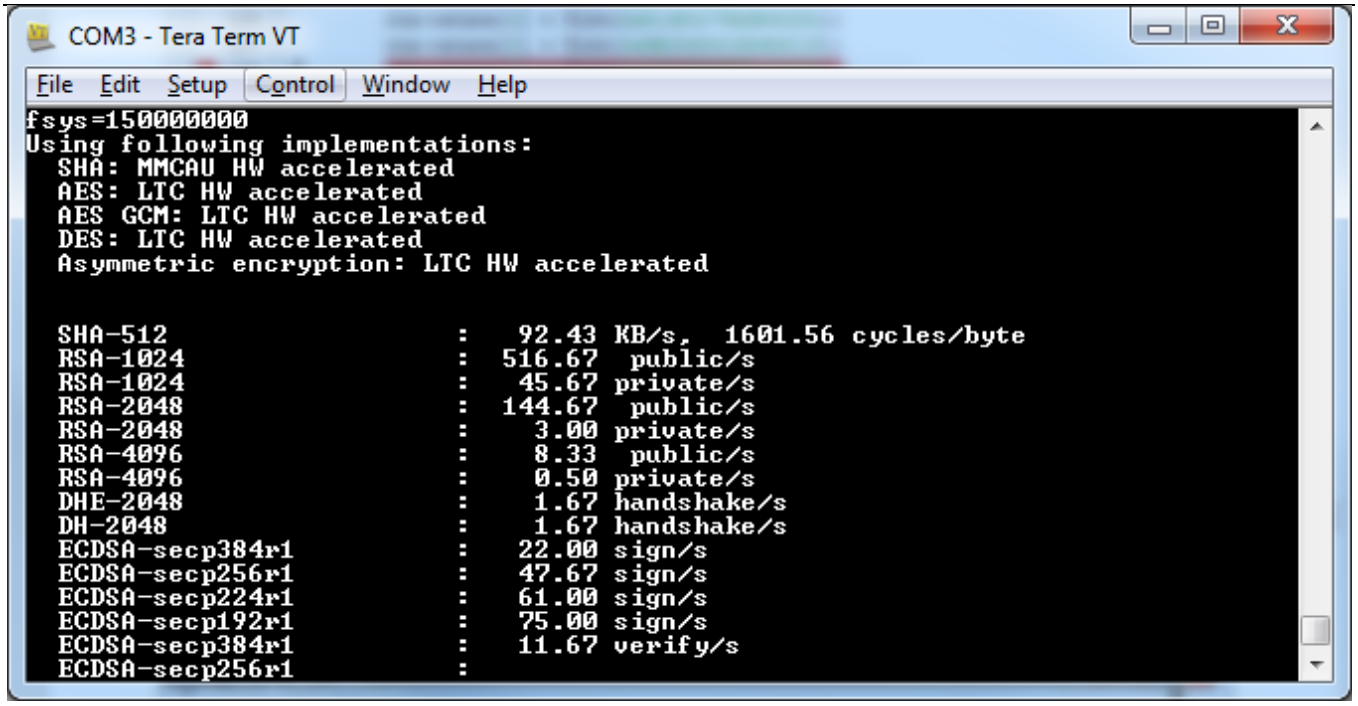


Return to SHA512.c and see that breakpoints can be set on line 128.

```
121          /* SHA-512 */
122          ctx->state[0] = UL64(0x6A09E667F3BCC908);
123          ctx->state[1] = UL64(0xBB67AE8584CAA73B);
124          ctx->state[2] = UL64(0x3C6EF372FE94F82B);
125          ctx->state[3] = UL64(0xA54FF53A5F1D36F1);
126          ctx->state[4] = UL64(0x510E527FADE682D1);
127          ctx->state[5] = UL64(0x9B05688C2B3E6C1F);
128          ctx->state[6] = UL64(0x1F83D9ABFB41BD6B);
129          ctx->state[7] = UL64(0x5BE0CD19137E2179);
130      }
```

Remove the breakpoint and run the program to see the new performance.



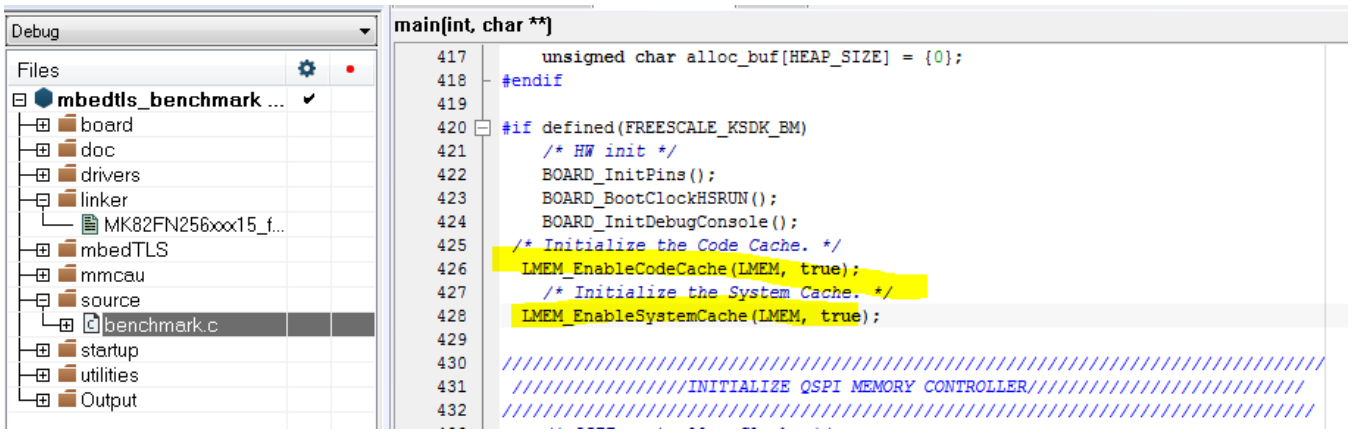


Performance improves!

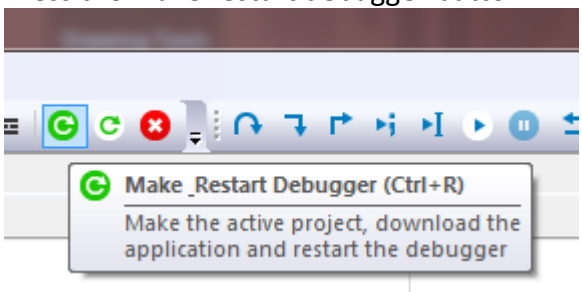
8 Enabling CPU Cache for best performance

Enable CPU Cache

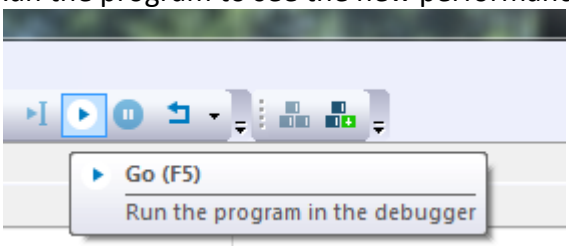
Open benchmark.c file and uncomment line 426 and line 428

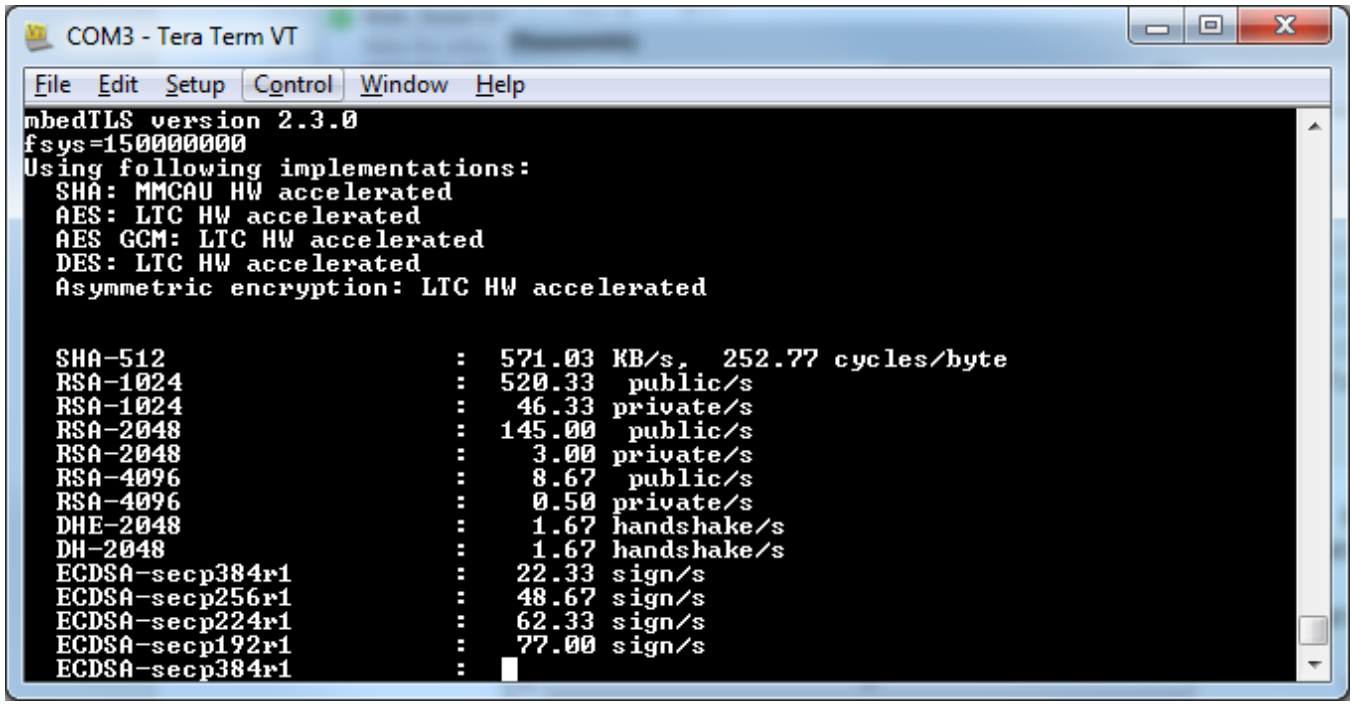


Press the make restart debugger button



Run the program to see the new performance.





9 Conclusion

Thanks for completing the QuadSPI Execute in place lab. Please explore the resources section.