

ADVANCED DEBUGGING WITH MCUXPRESSO IDE V11.1

AMF-SOL-T4020

MARCH 2020



PUBLIC



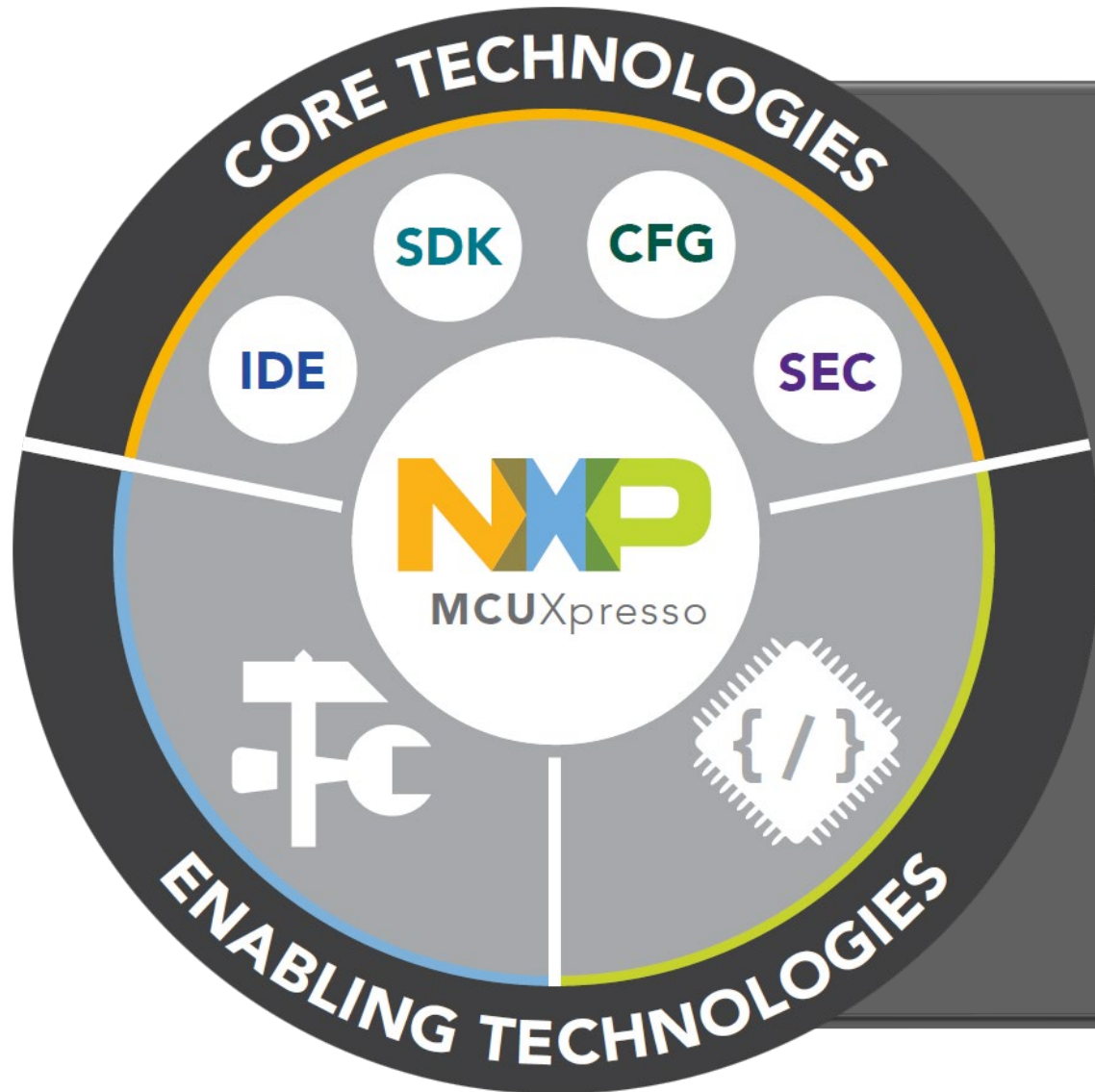
SECURE CONNECTIONS
FOR A SMARTER WORLD

OVERVIEW AND PRE-REQUISITES

Pre-requisites

- Experience of programming in C
- Watch the [MCUXpresso tool suite overview video](#) (<5 mins long)
- Using MCUXpresso SDK selection from within MCUXpresso IDE (7 mins long):
 - See this [video](#) on nxp.com
 - Using the tutorial as a guideline, install the SDK for the board you are using

The MCUXpresso Ecosystem



Core Technologies from NXP:

- MCUXpresso IDE
- MCUXpresso SDK
- MCUXpresso Config Tools
- MCUXpresso Secure Provisioning Tool

Enabling Software Technologies:

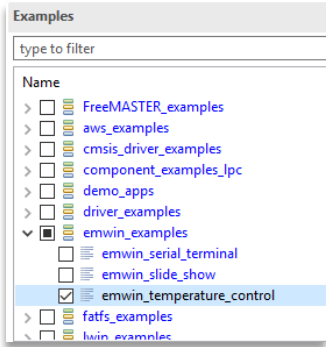
- Run time software libraries and middleware
- Enable customers to focus on differentiation
- From NXP and partners

Enabling Tools Technologies:

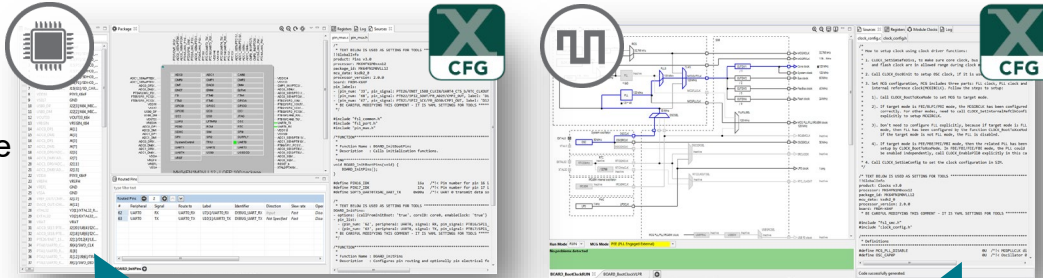
- Partner IDEs
- Debug Probes
- Development Boards
- From NXP and partners

Evaluation to proof of concept on NXP Evaluation Boards

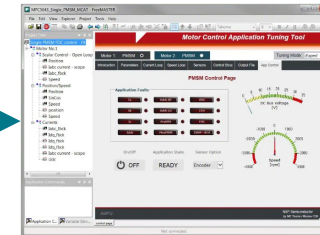
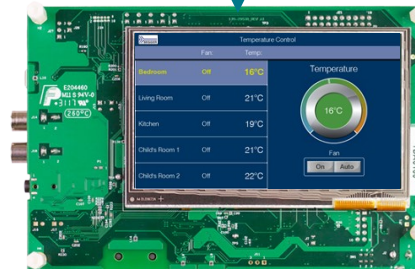
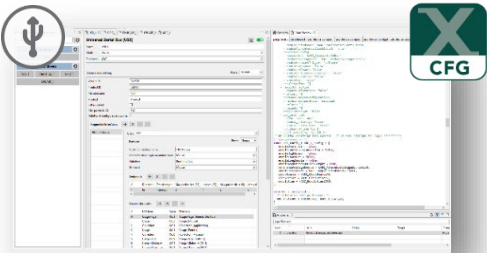
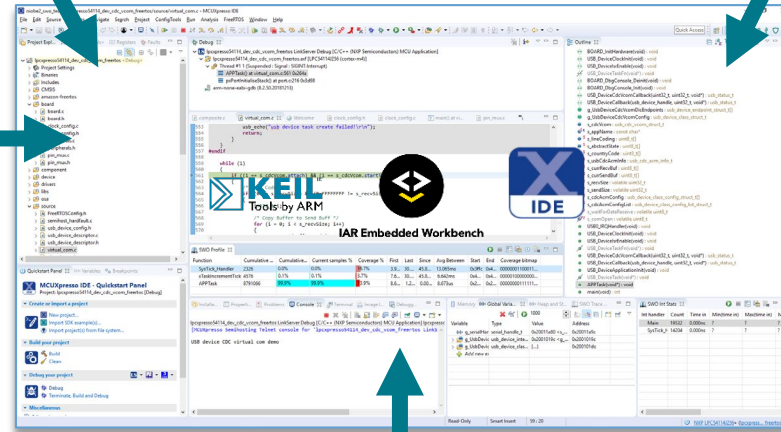
- Import/clone one of a large range of SDK examples
- Easy selection from with MCUXpresso IDE



- Or use MCUXpresso IDE New project wizard and Peripheral config tool to select and configure drivers and middleware



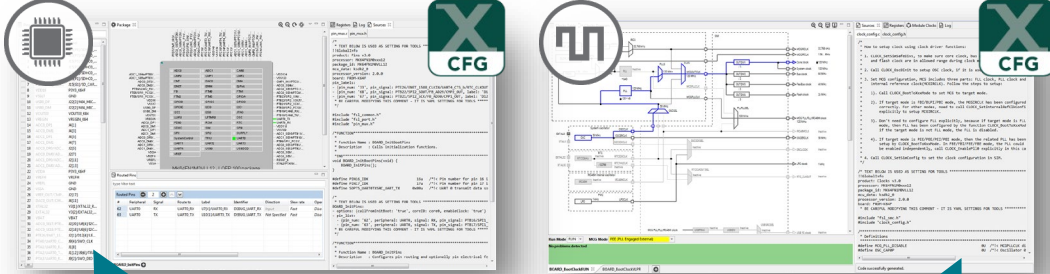
- Modify pin/clock settings for your application
- Simple, push button updates into IDE project



- Visualize data in real time
- Implement debug control interfaces

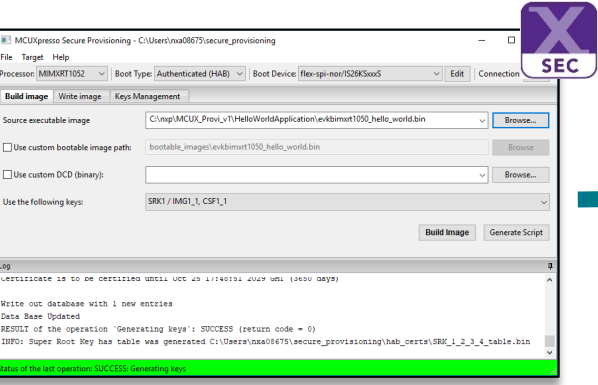
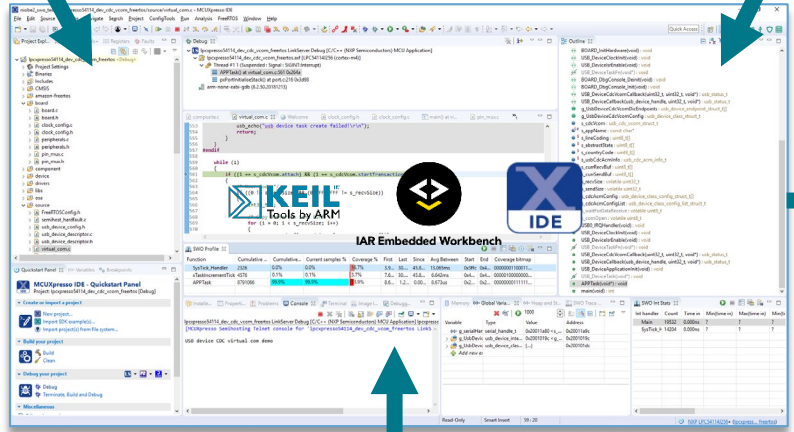


Transition to custom hardware and on to production

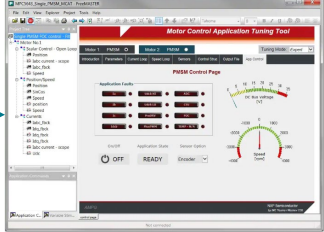


- Modify pin/clock settings for custom hardware
- Simply update then rebuild for custom board

- Encrypt/sign application images
- Setup provisioning and programming scripts



• Production



- Same visualization and control as on NXP evaluation board
- Options to choose different interfaces



Lab Setup/Prerequisites (FRDM-K64, if available)

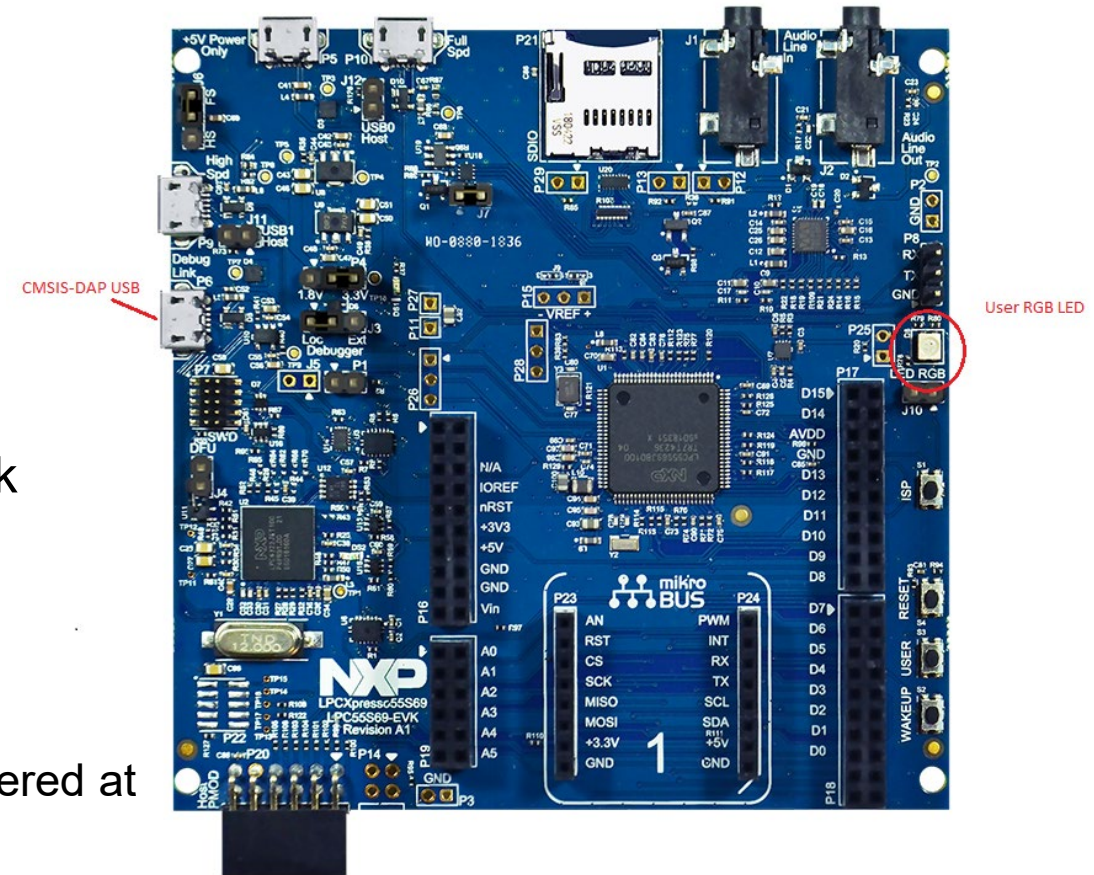
- MCUXpresso IDE 11.1.x
 - <https://www.nxp.com/mcuxpresso/ide>
- FRDM-K64 SDK V2.6.0 or later
 - <http://mcuxpresso.nxp.com/>
- FRDM-K64F Board with micro-USB Cable
 - <https://www.nxp.com/freedom>
- FRDM-K64F Board with DAPLink/CMSIS-DAP Firmware
 - Bootloader rev0244 OpenSDA v2.2
 - DAPLink rev0244 Firmware
 - <https://www.nxp.com/opensda>



```
DETAILS.TXT - Notepad
File Edit Format View Help
# DAPLink Firmware - see https://mbed.com/daplink
Unique ID: 0240000028884e450007700f6bf000278021000097969900
HIC ID: 97969900
Auto Reset: 0
Automation allowed: 0
Overflow detection: 0
Daplink Mode: Interface
Interface Version: 0244
Bootloader Version: 0244
Git SHA: 5f9092d41cfd6601fef7b3b467fe8f8767b01f84
Local Mods: 1
USB Interfaces: MSD, CDC, HID
Bootloader CRC: 0x251003d3
Interface CRC: 0x0676bc5d
Remount count: 0
```

Lab Setup/Prerequisites (LPCXpresso boards)

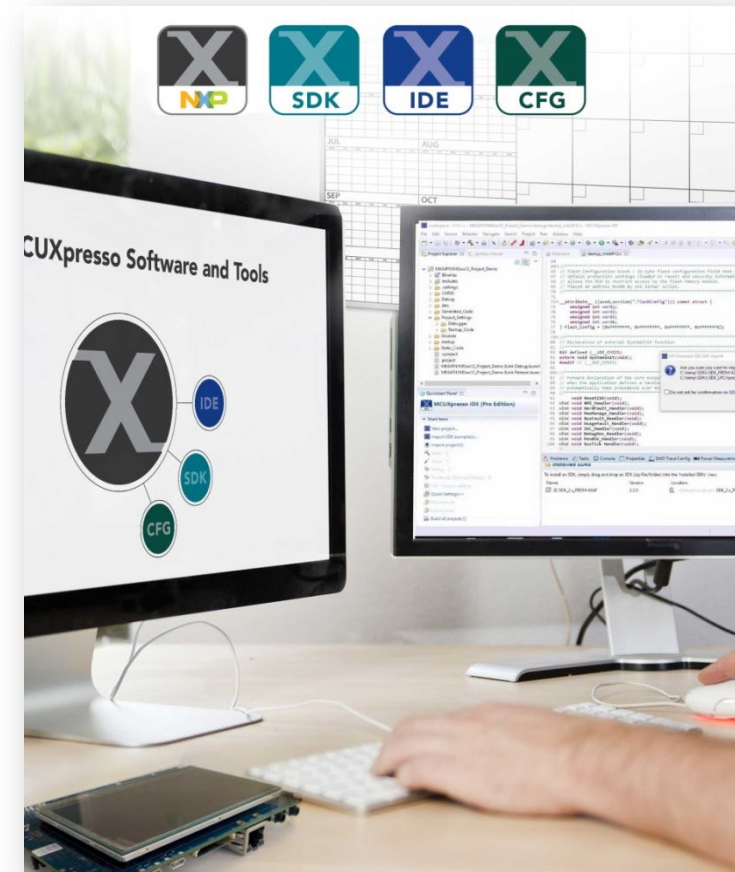
- Must be MCUXpresso SDK supported
 - LPC54xxx, LPC55xx, LPC51U68
- MCUXpresso IDE 11.1.1
 - <https://www.nxp.com/mcuxpresso/ide>
- SDK V2.7.x or later for board being used
 - <http://mcuxpresso.nxp.com/>
- Board with micro-USB Cable (e.g. ...)
 - <https://www.nxp.com/demoboard/LPC55S69-EVK>
- Board may be pre-programmed with CMSIS-DAP or J-Link firmware, but not essential
- LPC8xx boards may also be used
 - Other boards above are a better option due SWO support
 - Will have LPC11U35, CMSIS-DAP debug probe
 - Ensure version 1.0.7 or later (will be shown when probe discovered at start of debug session)



Advanced Debug Course Sections

Introduction/pre-requisites

1. **Part 1: Basic debugging and code flashing**
 - **Creating/Cloning MCUXpresso SDK Projects in the IDE**
 - **Building, Basic Debugging**
 - **Startup, Connect, Disconnect, Attach**
 - **GUI Flash Tool**
2. Part 2: Accessing data and peripherals
 - Global Variables, Variable Plots, Data Stack, Heap and Peripherals
 - Hard faults
3. Part 3: Halting execution
 - Breakpoints & Watchpoints
4. Part 4: Instruction trace*
5. Part 5: FreeRTOS Task Aware Debug
6. Part 6: SWO trace, profiling and ITM**



*Instruction trace requires Cortex M0+ with MTB support or Cortex M4 w/ ETM

**SWO profiling requires LPC55xx/54xxx/51U68, i.MX RT1060 or i.MX RT10x0 with LPC-Link2 probe

PART 1: BUILDING, DEBUGGING AND DIRECT FLASHING

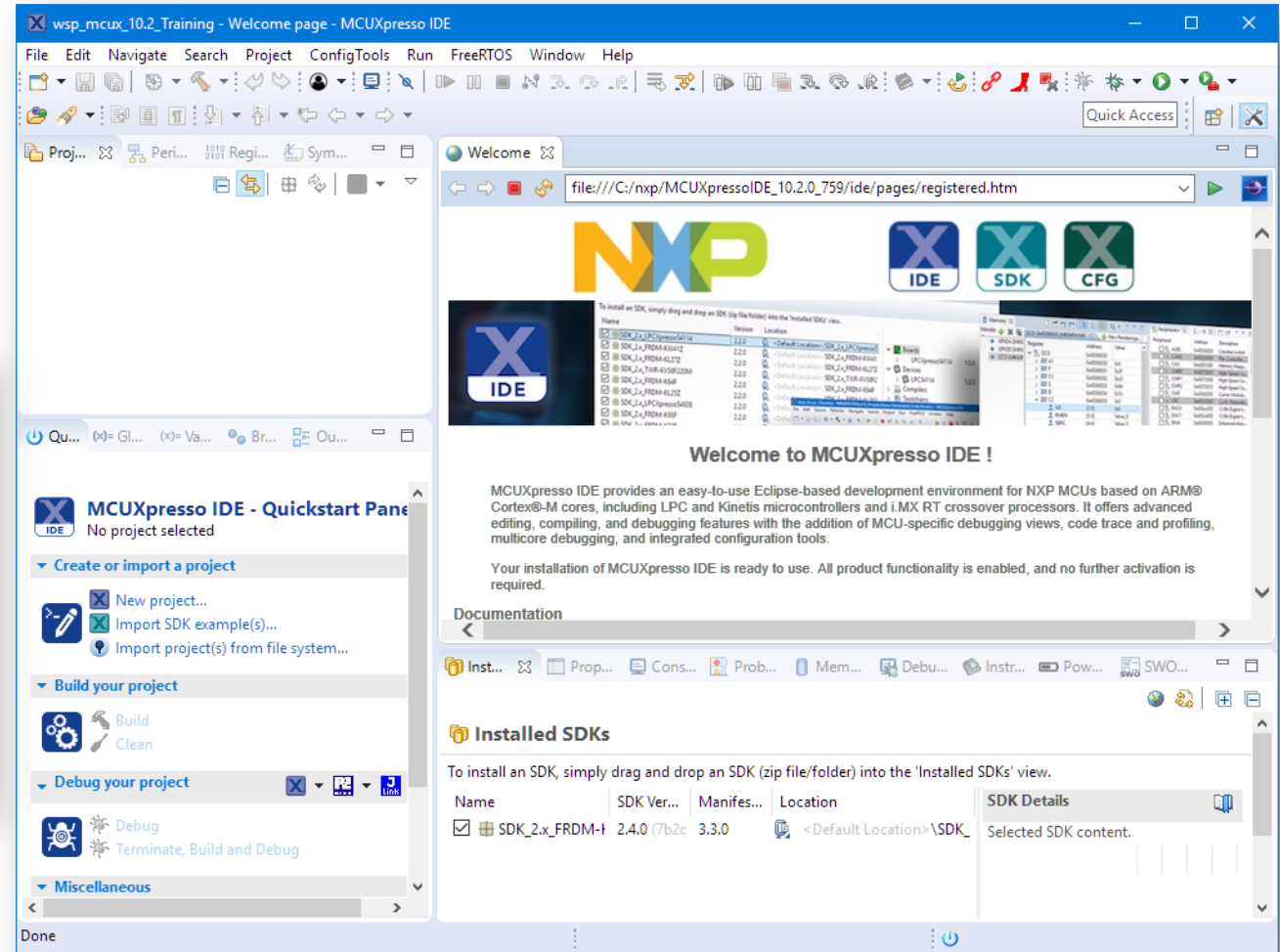
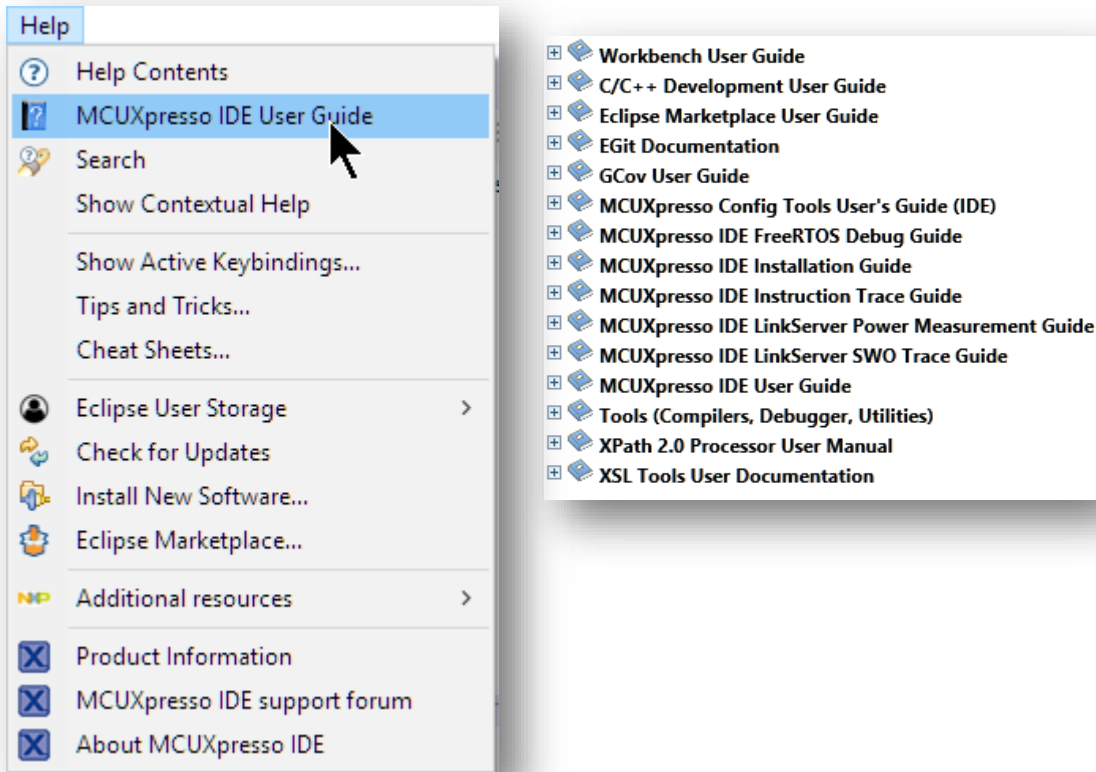


About this tutorial and the board you are using

- All steps in the tutorial are the same for whatever board is being used, but debug probes vary between boards
 - FRDM boards use micro USB to OpenSDA, as do some i.MX RT 4-digit boards
 - LPC54000 or LPC5500 series will have LPC-Link2 CMSIS-DAP probe
 - LPC800 boards will have LPC11U35 CMSIS-DAP probe
- The IDE debug system discovers probe for any of these, but the probe name will be different

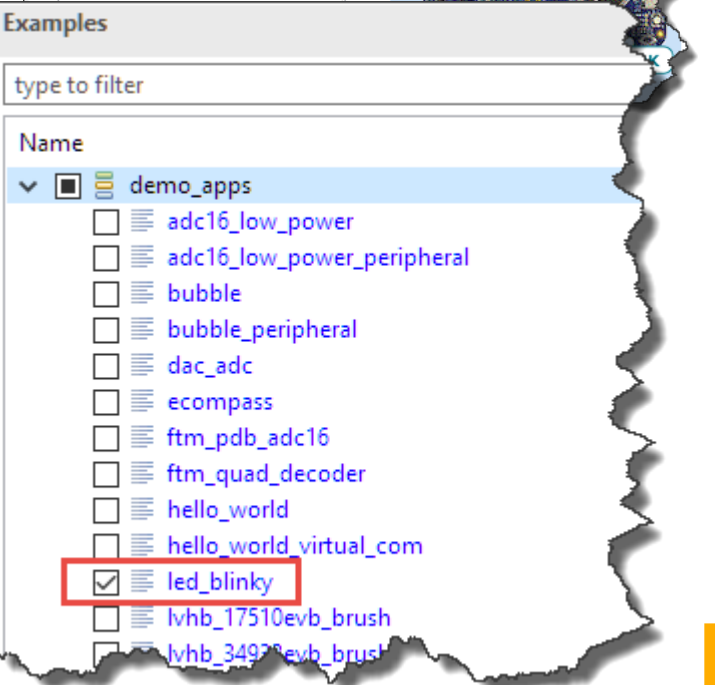
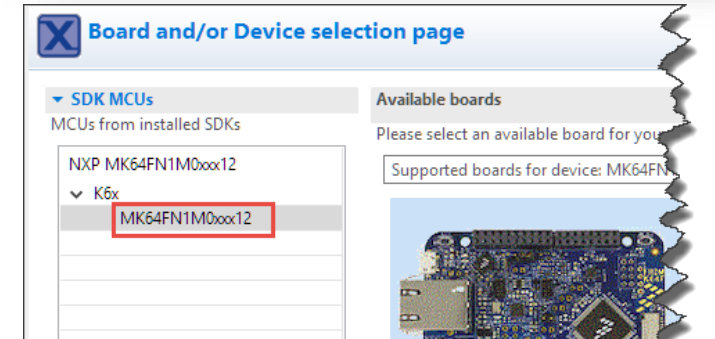
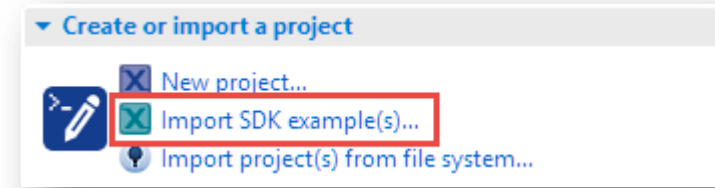
MCUXpresso IDE

- Start the IDE with Shortcut
- Select workspace
- Open MCUXpresso IDE User Guide



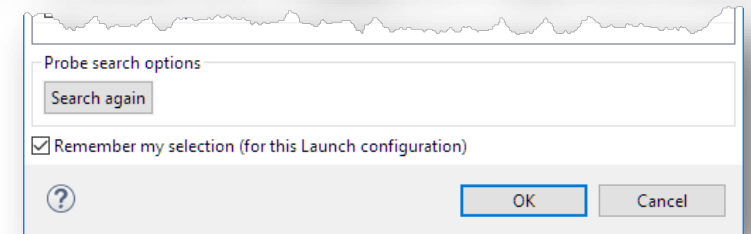
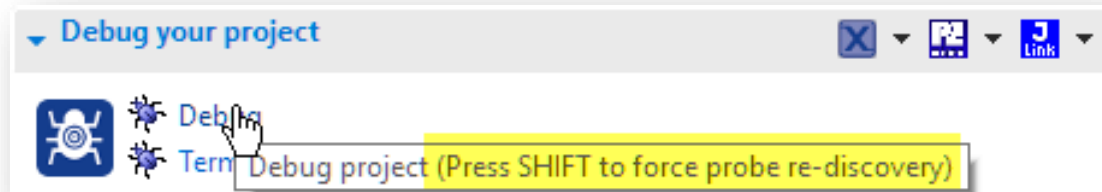
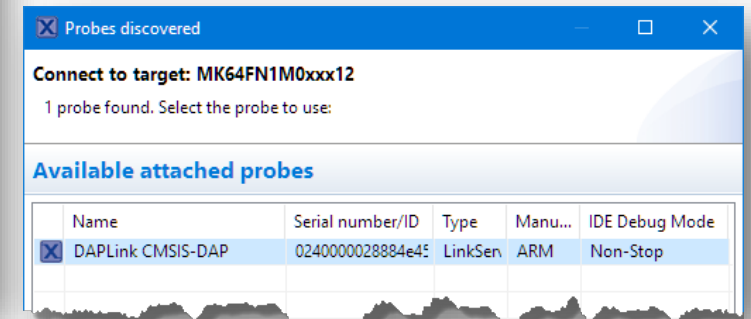
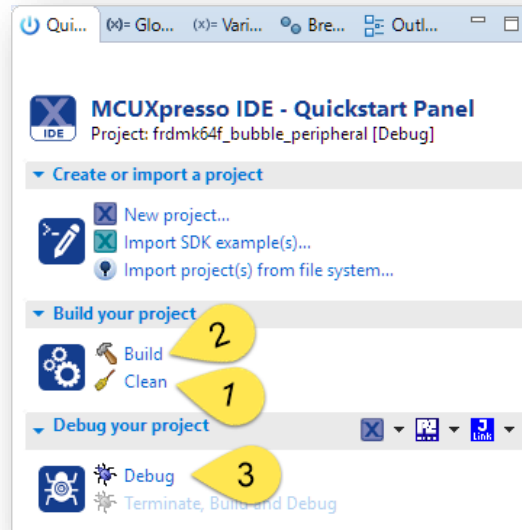
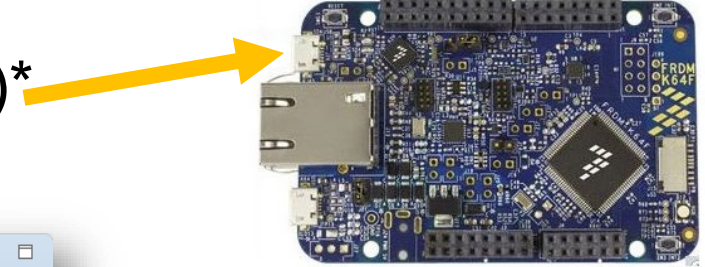
Create LED Blinky Project

- Prerequisite:
 - SDK_2.x_FRDM-K64F (or SDK for the board being used) installed in IDE
- Use Quickstart Panel with “Import SDK example(s)...”
 - Select frdmk64f board image (or board being used)
 - Click “Next”
 - Select “demo_apps > led_blinky”
 - Click “Finish”

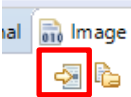


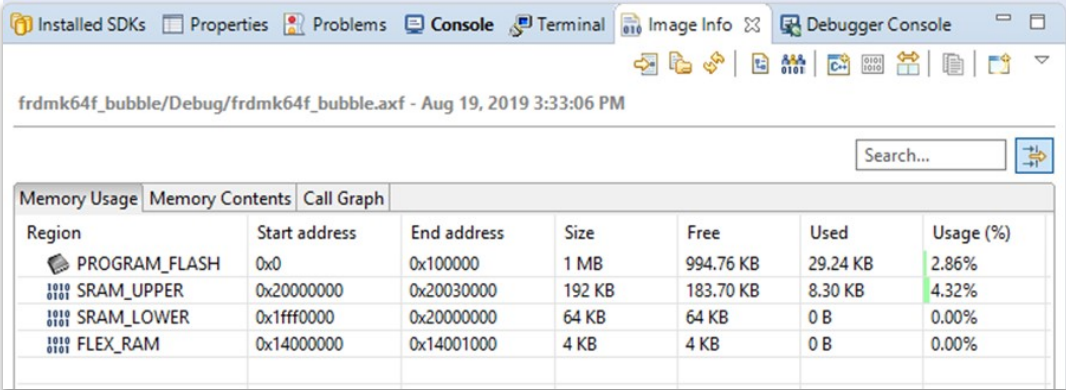
Build and Debug (FRDM-K64F)

- **Connect FRDM-K64F board** (micro USB to OpenSDA)*
 - Windows may need to enumerate USB connection
- **Use IDE Quickstart Panel to:**
 - **Clean**
 - **Build**
 - **Debug**
- **Debugger discovers probe**
 - Use **SHIFT** to force probe re-discovery

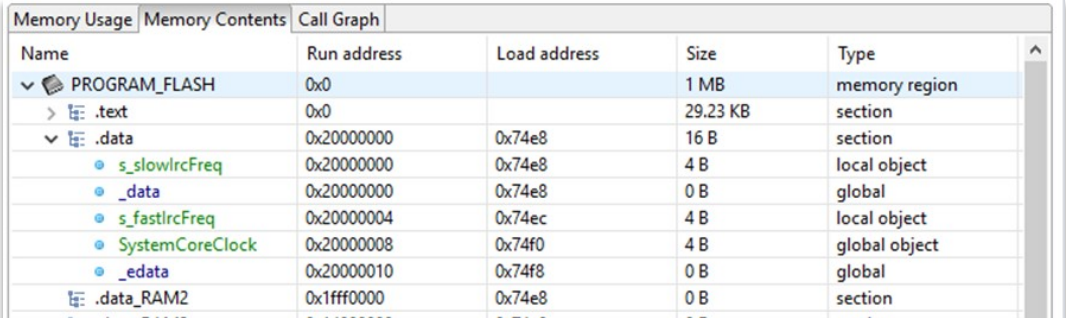


Debug Info dialog

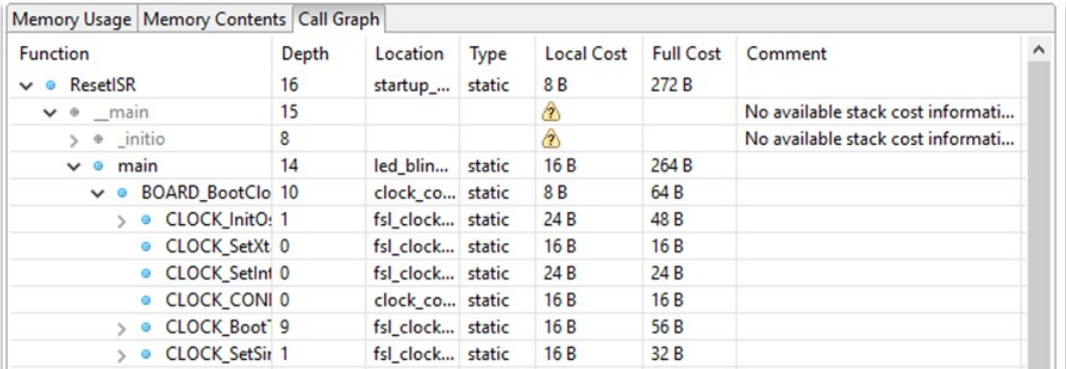
- Image info provides a range of information on a built application
- Must load image info after selecting project 
- Memory usage: overall memory use by region
- Memory contents: detailed breakdown of all symbols by section
- Call graph: shows functions called by each function, stack usage (“Cost”)
 - Useful for code size optimization



Region	Start address	End address	Size	Free	Used	Usage (%)
PROGRAM_FLASH	0x0	0x1000000	1 MB	994.76 KB	29.24 KB	2.86%
SRAM_UPPER	0x20000000	0x20030000	192 KB	183.70 KB	8.30 KB	4.32%
SRAM_LOWER	0x1fff0000	0x20000000	64 KB	64 KB	0 B	0.00%
FLEX_RAM	0x14000000	0x14001000	4 KB	4 KB	0 B	0.00%



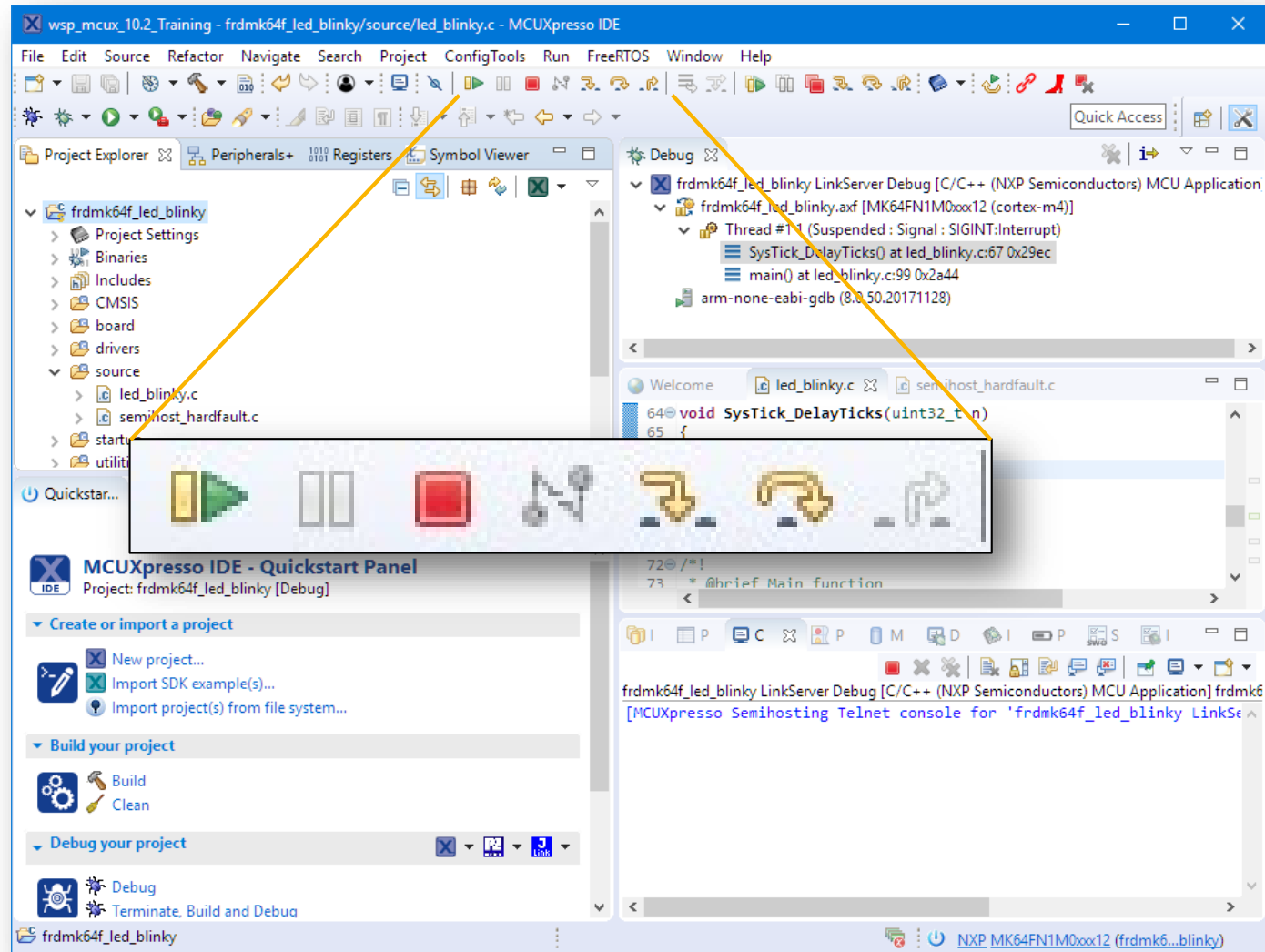
Name	Run address	Load address	Size	Type
PROGRAM_FLASH	0x0		1 MB	memory region
.text	0x0		29.23 KB	section
.data	0x20000000	0x74e8	16 B	section
s_slowlrcFreq	0x20000000	0x74e8	4 B	local object
_data	0x20000000	0x74e8	0 B	global
s_fastlrcFreq	0x20000004	0x74ec	4 B	local object
SystemCoreClock	0x20000008	0x74f0	4 B	global object
_edata	0x20000010	0x74f8	0 B	global
.data_RAM2	0x1fff0000	0x74e8	0 B	section



Function	Depth	Location	Type	Local Cost	Full Cost	Comment
ResetISR	16	startup_...	static	8 B	272 B	
_main	15			?		No available stack cost informati...
_initio	8			?		No available stack cost informati...
main	14	led_blin...	static	16 B	264 B	
BOARD_BootClo	10	clock_co...	static	8 B	64 B	
CLOCK_InitO: 1		fsl_clock...	static	24 B	48 B	
CLOCK_SetXt 0		fsl_clock...	static	16 B	16 B	
CLOCK_SetIn 0		fsl_clock...	static	24 B	24 B	
CLOCK_CONI 0		clock_co...	static	16 B	16 B	
CLOCK_Boot' 9		fsl_clock...	static	16 B	56 B	
CLOCK_SetSir 1		fsl_clock...	static	16 B	32 B	

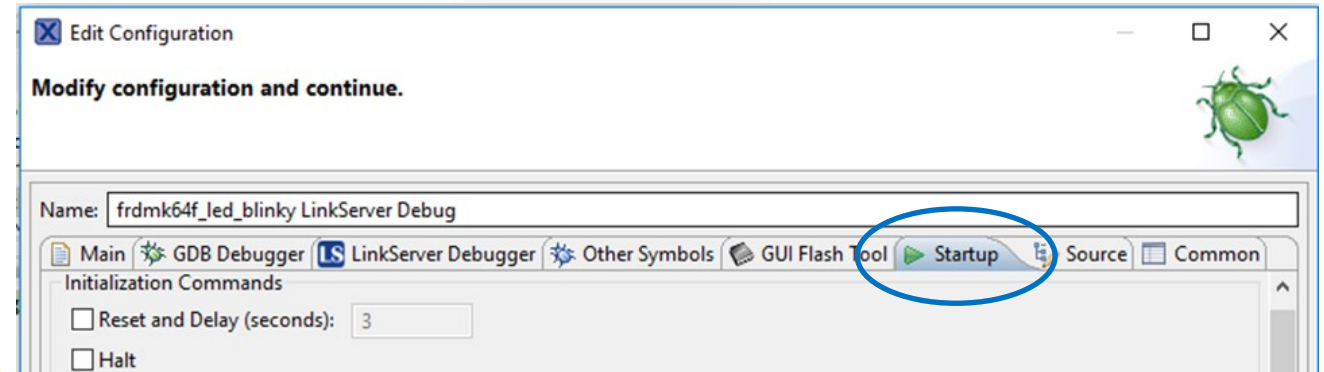
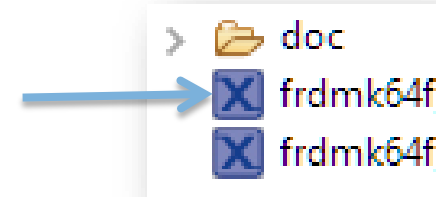
Debugger Run Control

- Resume/Run
- Suspend/Pause
- Terminate/Stop
- (Disconnect)
- Step Into
- Step Over
- Step Out
- Step through the code



Debug Startup Breakpoint

- By default, target runs until main()
- Double-Click on Debug *.launch File in Project
 - Opens Debug Configuration
 - Stop at **ResetISR**
- **Debug**
- Can now debug startup code

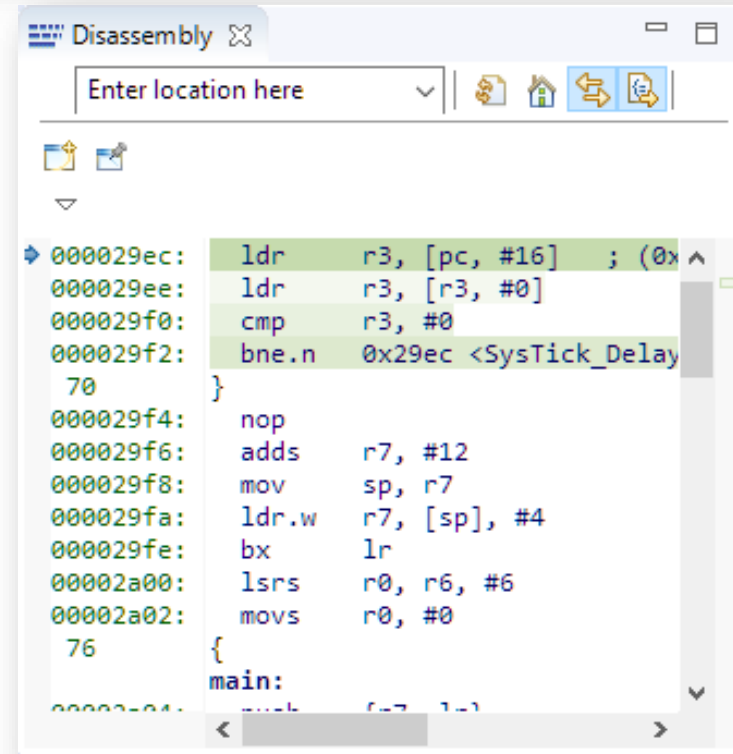
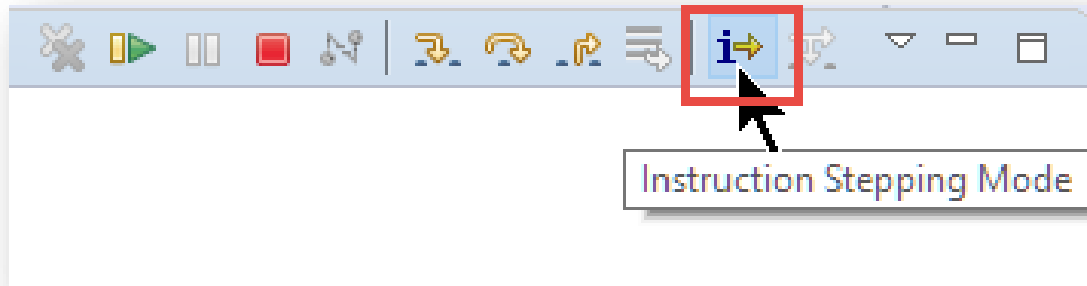
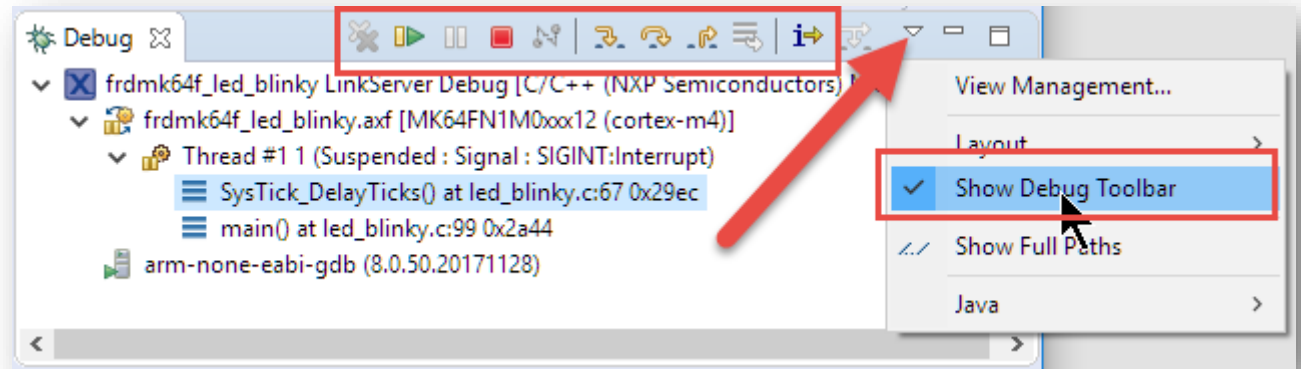


```
startup_mk64f12.c
455 //*****
456 // Reset entry point for your code.
457 // Sets up a simple runtime environment and initializes t
458 // library.
459 //*****
460 __attribute__((section(".after_vectors.reset")))
461 void ResetISR(void) {
462
463     // Disable interrupts
464     __asm volatile ("cpsid i");
465
466     #if defined (__USE_CMSIS)
467     // If __USE_CMSIS defined, then call CMSIS SystemInit code
468     SystemInit();
469
470 #else
471     // Disable Watchdog
```



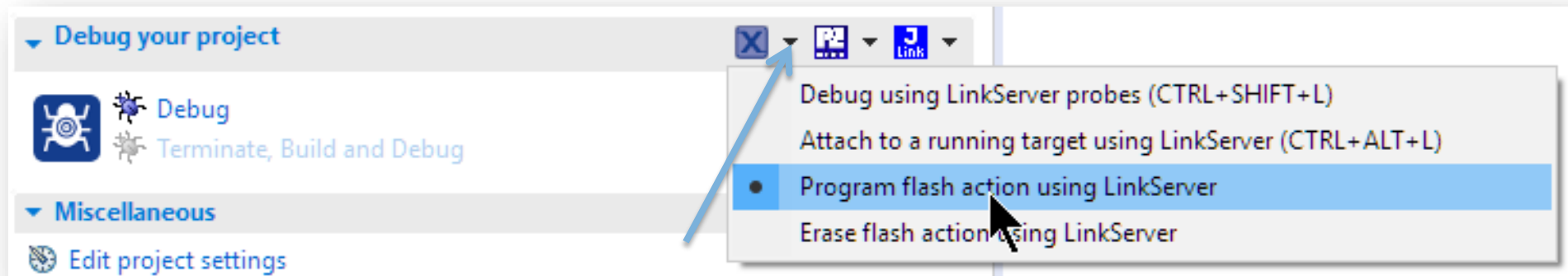
Debug Toolbar in Debug View, Assembly Stepping

- Enable debug toolbar
- Turn on instruction stepping
- Perform stepping
- Switch back to Source stepping



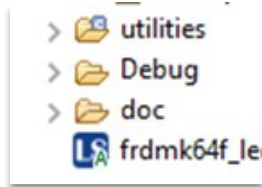
Debug Quickstart Shortcuts

- Quick and fast way to
 - Debug (default)
 - Attach
 - Program
 - Erase
- Setting is persistent between sessions

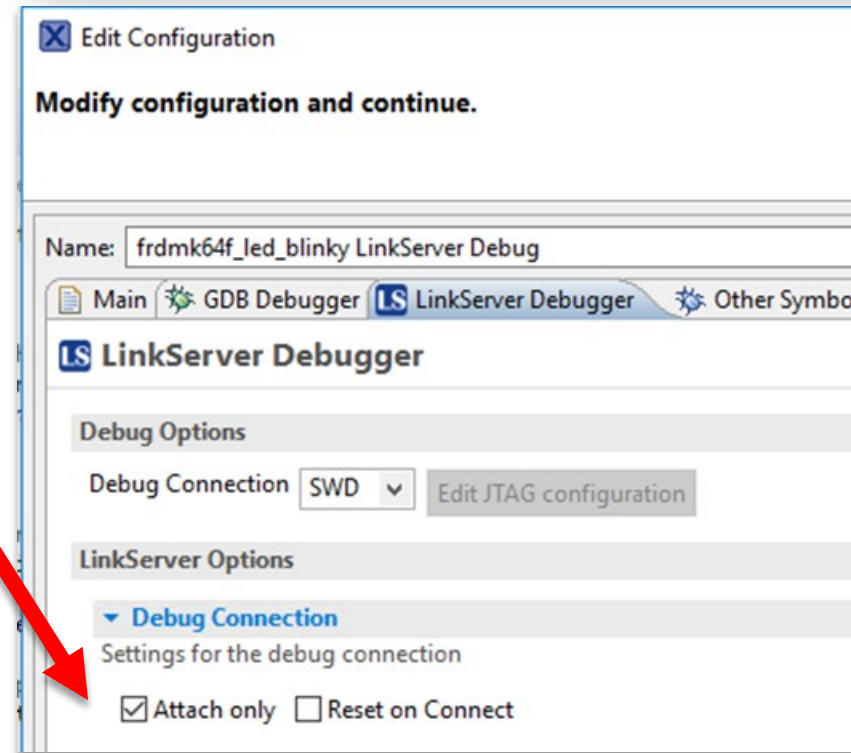
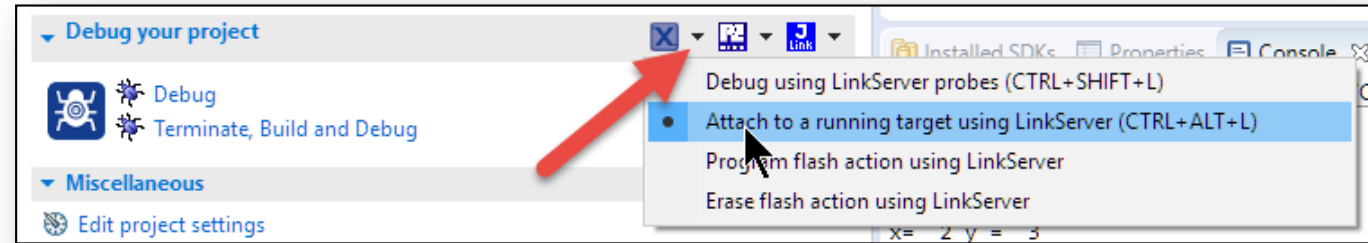


Debug Quickstart Shortcuts: Attach

- Use 'Attach' to running target
- Launch Configuration Icon has 'A' Decorator added:

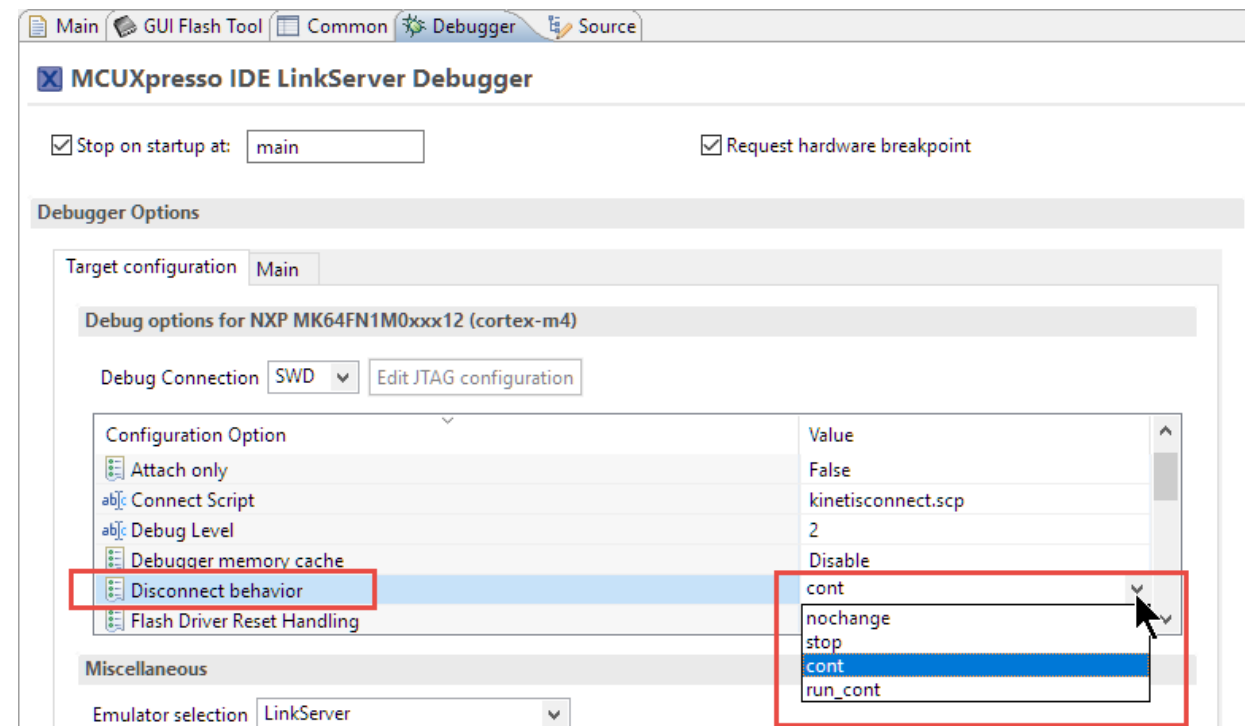
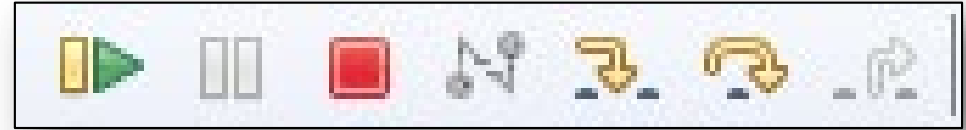


- Subsequent use of Debug (instead of Attach) will trigger a prompt to confirm reversion to Debug
- Using Attach option will result in persistent Setting in Debug Configuration
 - Will revert automatically in future releases



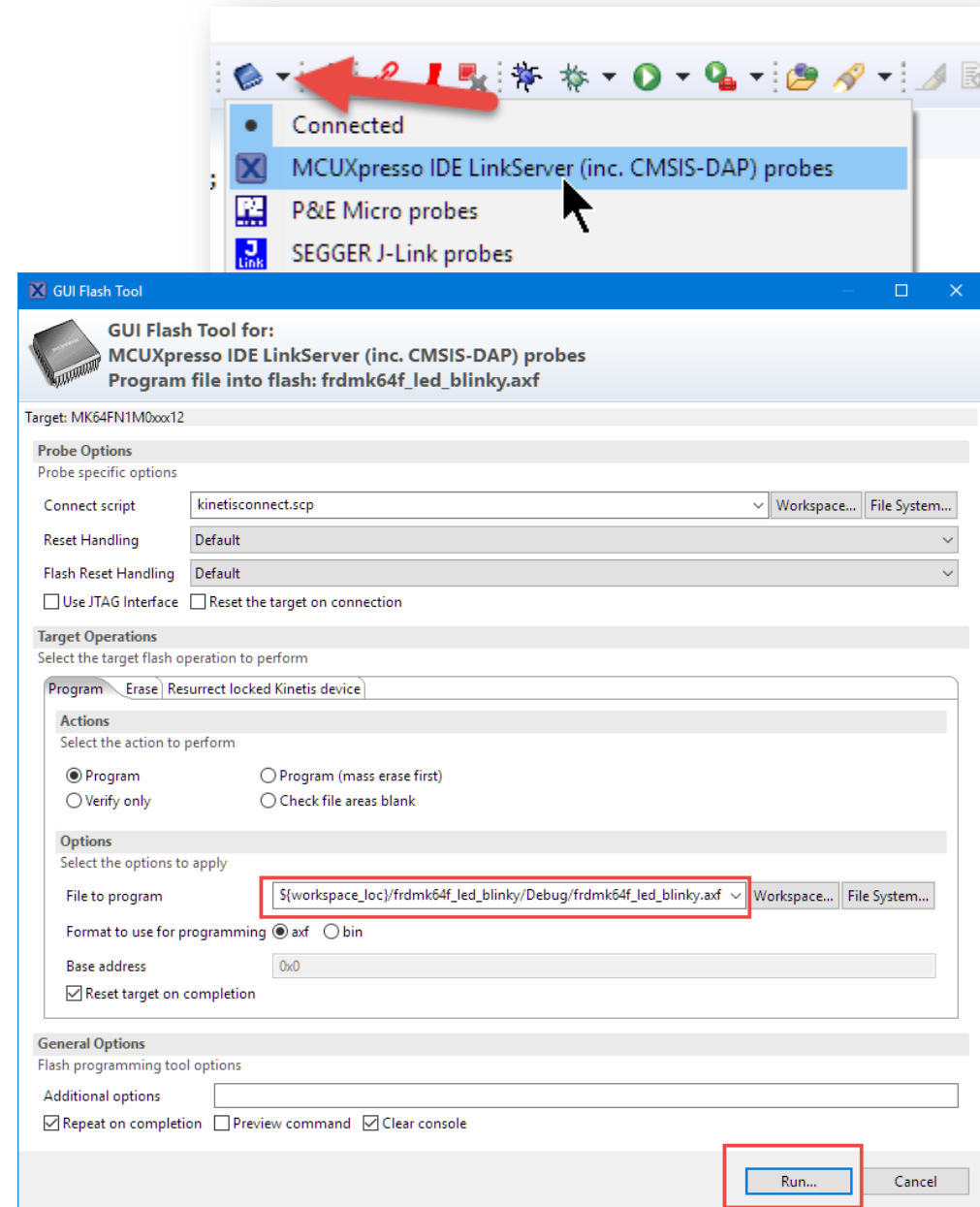
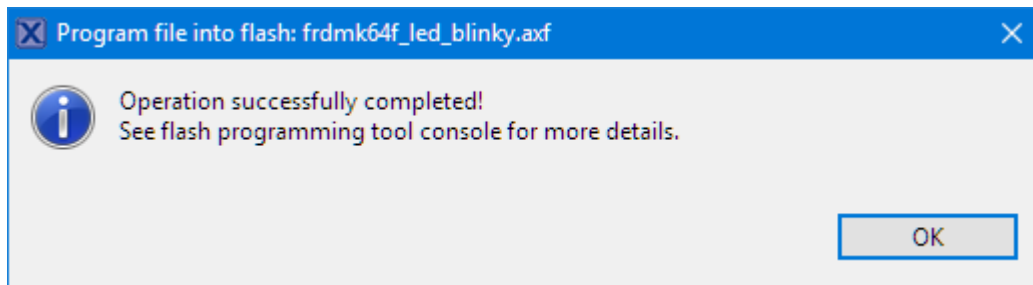
Disconnect

- 'Terminate'/Stop button disconnects from Target
- Open Debug Configuration
 - Double Click on *.launch file in Project
 - **nochange** - will leave the target in its current state
 - **stop** - will leave the target in debug state i.e. halted
 - **cont** - the default, will either start the image from its current PC value or leave it running
 - **run cont** - will reset the target and let it run
- Change it to 'stop' and try it



GUI Flash Tool

- Advanced Board flashing/programming
 - Program, verify, erase, check, resurrect, ...
 - Programming multiple boards
- Select **GUI Flash Tool** Icon
- **Run** Job
- Messages written to Console View
- **Check Console View**
- Can also be used for Mass Erase



PART 2: ACCESSING DATA AND PERIPHERALS

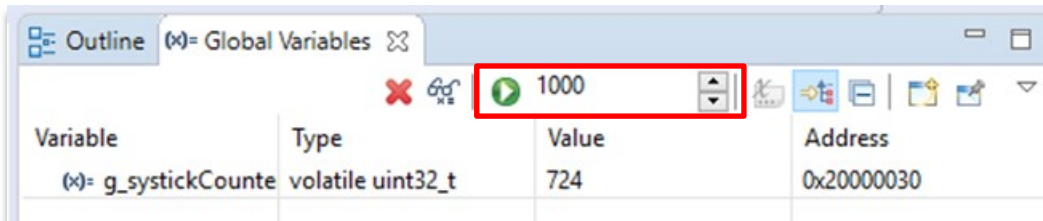
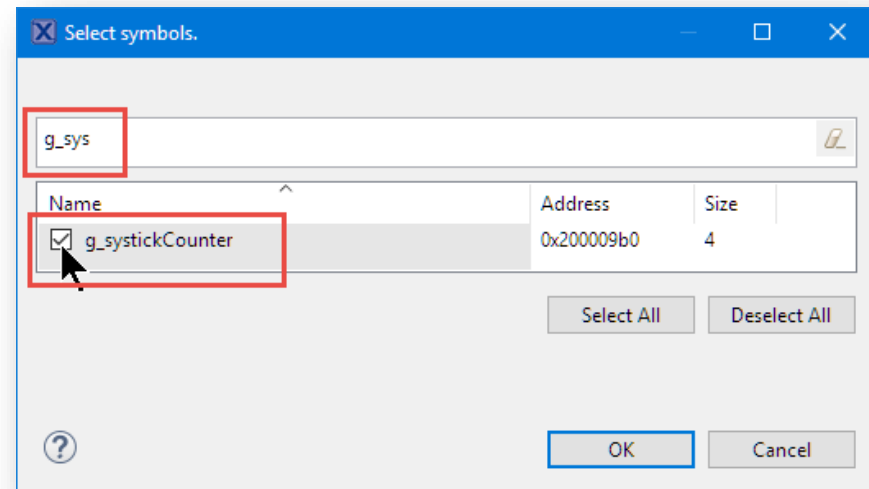
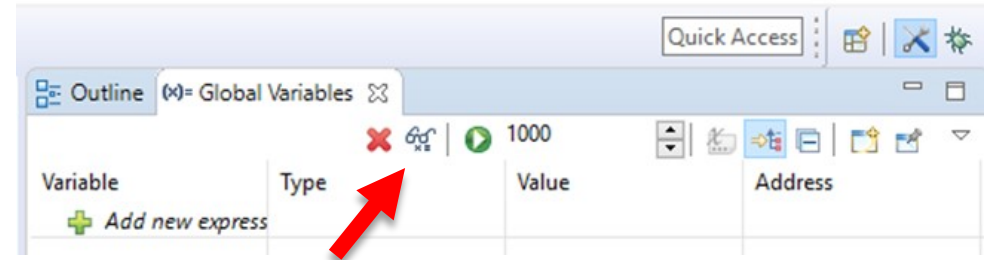


About this section

- Screen shots shown are from a session with FRDM-K64F
- All operations also apply to any MCUXpresso SDK supported board

Global Variables View Update

- Debug project
- Global Variables View
 - Add global variables
 - can use Filter for easier search, press OK
- Run target
- Variable is automatically updated
 - Refresh rate defaults to every 1000ms



Graphing variables

- Choose an example with data varying over time
 - Examples in this tutorial use the “bubble” application example with FRDM-K64F
 - Other options:
 - emWin touch and draw example on boards with LCD panel (e.g. LPC546xx, i.MX RT1050)
 - Any example reading on-board accelerometer or temperature sensor
 - Use blinky application and monitor the systick variable

Global variables: select symbols

- In variables dialog, click on the eyeglasses logo to open symbol selection dialog
 - If you cant find the global variables dialog, type “global” in the Quick access box
- Use filter to narrow down list of available globals (example from bubble application)

The image shows two screenshots of the Global Variables dialog and a Select symbols dialog. The top screenshot shows the Global Variables dialog with a table of variables. The bottom screenshot shows the Global Variables dialog with the checkboxes for g_xAngle and g_yAngle circled in red. The Select symbols dialog is open, showing a list of symbols with checkboxes and columns Name, Address, and Size. Red arrows point from the Select symbols dialog to the Global Variables dialog, highlighting the checkboxes for g_xAngle and g_yAngle.

Variable	Type	Value	Address
<input type="checkbox"/> + Add new expres			

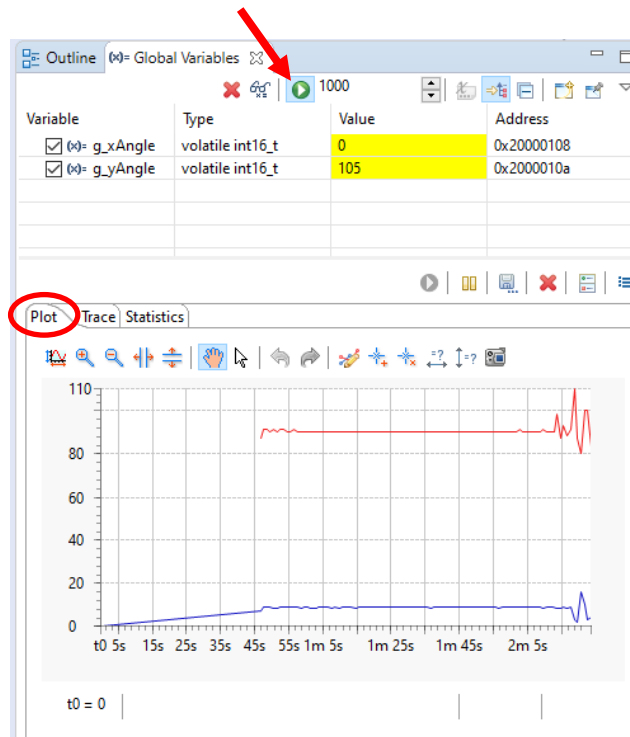
Name	Address	Size
<input type="checkbox"/> g_accel_address	0x000058b8	4
<input type="checkbox"/> g_pfnVectors	0x00000000	408
<input type="checkbox"/> g_serialHandle	0x2000002c	4
<input checked="" type="checkbox"/> g_xAngle	0x20000128	2
<input type="checkbox"/> g_xtal0Freq	0x20000030	4
<input type="checkbox"/> g_xtal32Freq	0x20000034	4
<input checked="" type="checkbox"/> g_yAngle	0x2000012a	2
<input type="checkbox"/> mcgConfig_BOARD_BootClockRUN	0x00006090	11
<input type="checkbox"/> oscConfig_BOARD_BootClockRUN	0x000060a4	8
<input type="checkbox"/> simConfig_BOARD_BootClockRUN	0x0000609c	8

Variable	Type	Value	Address
<input type="checkbox"/> + Add new expres			
<input type="checkbox"/> g_xAngle	volatile int16_t	0	0x20000128
<input type="checkbox"/> g_yAngle	volatile int16_t	0	0x2000012a

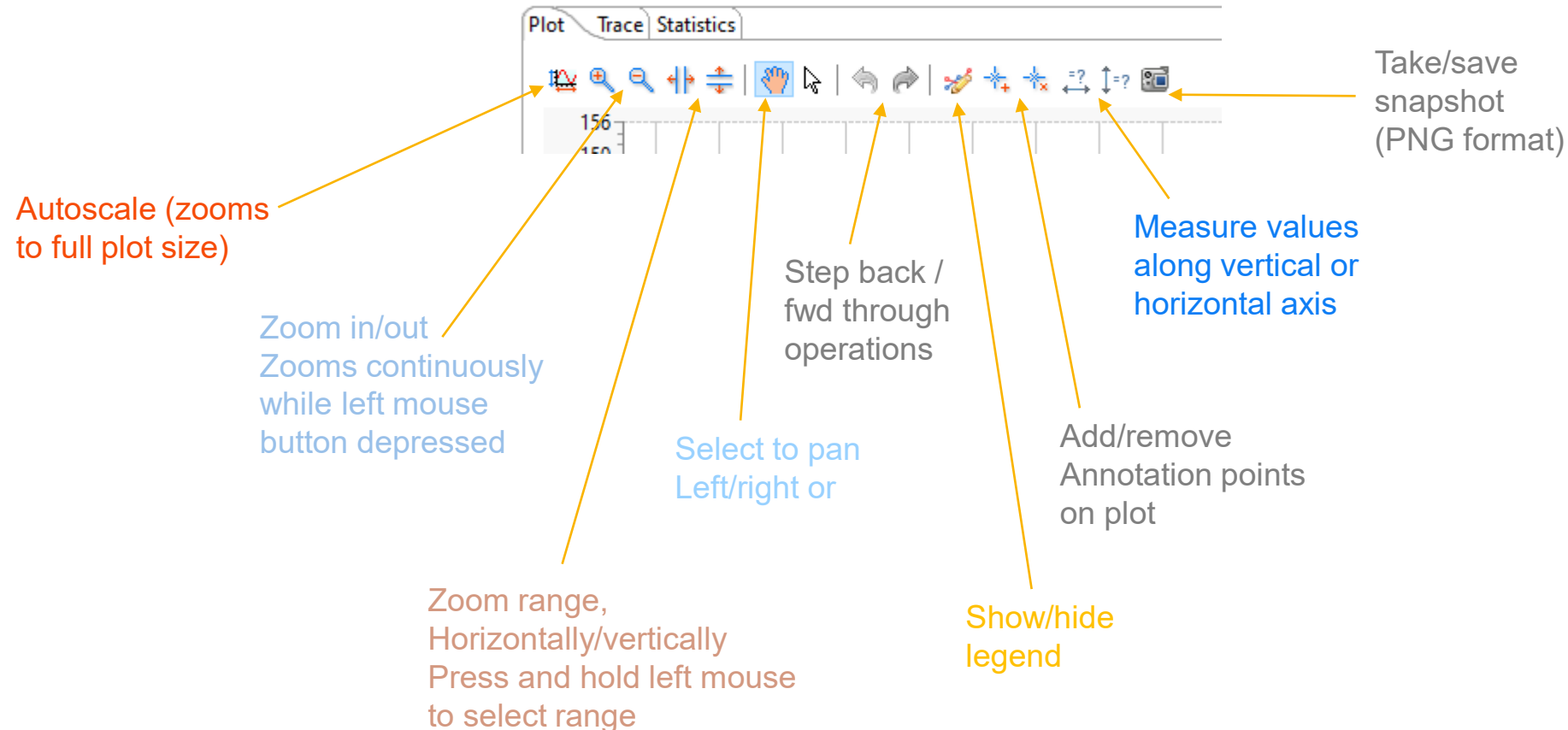
Check these boxes to enable graphing of these variables

Global variables: live value updates

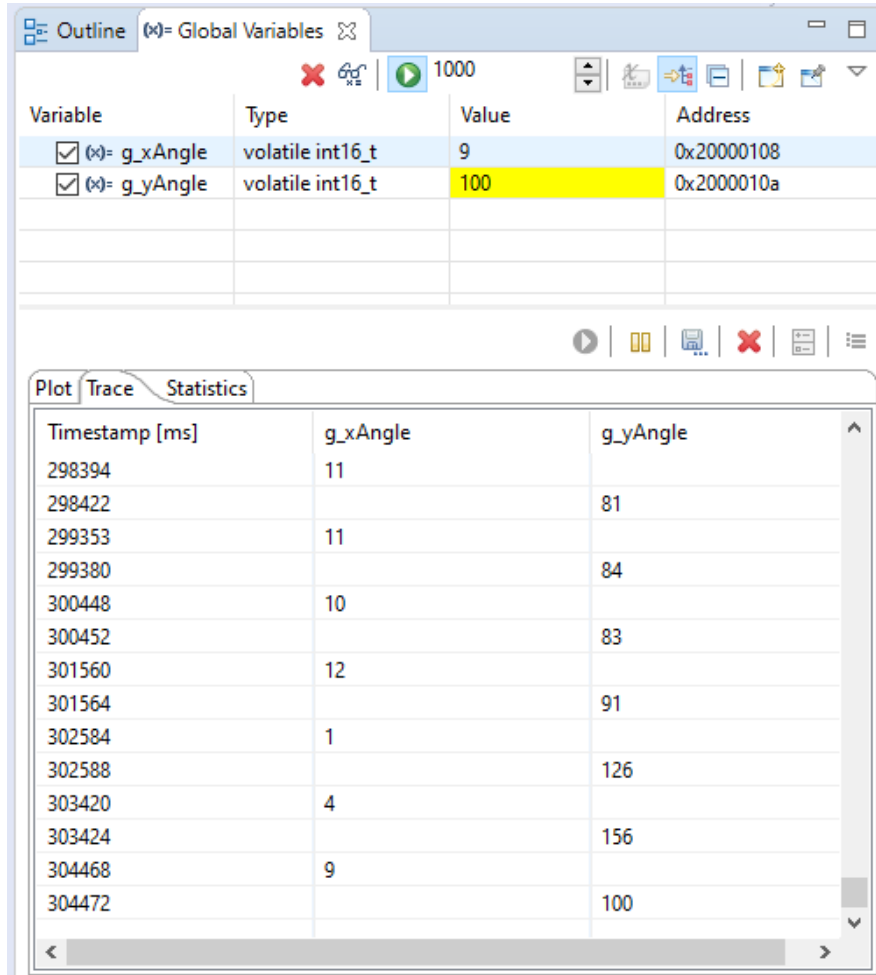
- Press green “play” symbol to start live polling of variables
- With bubble application running, move the board around to see values change
 - Changed variables highlight yellow, variables begin plotting in Plot tab



Global variables: Plotting controls

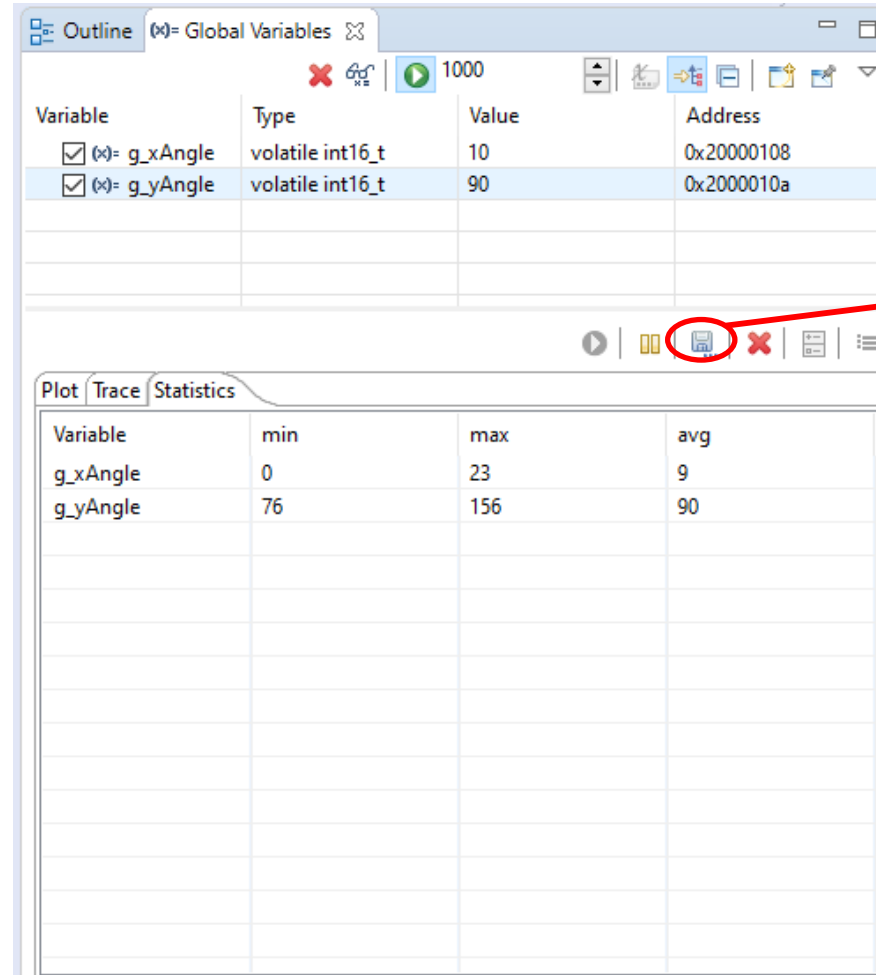


Global variables: Trace and statistics tabs



Global Variables window showing two variables: g_xAngle (value 9) and g_yAngle (value 100). The Trace tab is active in the bottom panel, showing a table of values over time.

Timestamp [ms]	g_xAngle	g_yAngle
298394	11	
298422		81
299353	11	
299380		84
300448	10	
300452		83
301560	12	
301564		91
302584	1	
302588		126
303420	4	
303424		156
304468	9	
304472		100



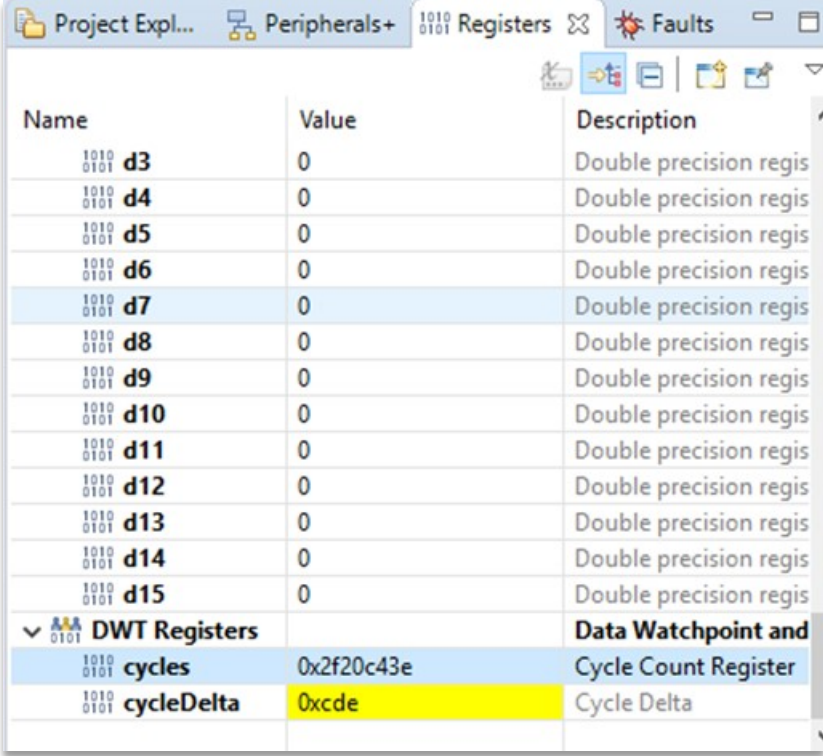
Global Variables window showing two variables: g_xAngle (value 10) and g_yAngle (value 90). The Statistics tab is active in the bottom panel, showing a table of min, max, and avg values for each variable. A red circle highlights the save icon in the toolbar.

Variable	min	max	avg
g_xAngle	0	23	9
g_yAngle	76	156	90

Clicking the save icon with any of the tabs active will result in dialogs to save PNG file for plot .tsv files for Trace and Statistics

Registers View

- Registers View for core registers
- If view not already open: Menu *Window* > *Show View* > *Other...* > *Debug* > *Registers*
- Pseudo Registers for Cycle Counters
 - **cycleDelta**: cycles since the last CPU stop
- Measure time of code execution between two breakpoints



Name	Value	Description
<small>1010 0101</small> d3	0	Double precision regis
<small>1010 0101</small> d4	0	Double precision regis
<small>1010 0101</small> d5	0	Double precision regis
<small>1010 0101</small> d6	0	Double precision regis
<small>1010 0101</small> d7	0	Double precision regis
<small>1010 0101</small> d8	0	Double precision regis
<small>1010 0101</small> d9	0	Double precision regis
<small>1010 0101</small> d10	0	Double precision regis
<small>1010 0101</small> d11	0	Double precision regis
<small>1010 0101</small> d12	0	Double precision regis
<small>1010 0101</small> d13	0	Double precision regis
<small>1010 0101</small> d14	0	Double precision regis
<small>1010 0101</small> d15	0	Double precision regis
<small>1010 0101</small> DWT Registers		Data Watchpoint and
<small>1010 0101</small> cycles	0x2f20c43e	Cycle Count Register
<small>1010 0101</small> cycleDelta	0xcde	Cycle Delta

Hard Fault – Faults view

- Add code writing to read-only memory

```
59 BOARD_InitPins();
60 BOARD_BootClockRUN();
61 BOARD_InitDebugConsole();
62
63 /* Init output LED GPIO. */
64 GPIO_PinInit(BOARD_LED_GPIO, BOARD_LED_GPIO_PIN, &led_config);
65 *(int*)0 = 0;
66
67 /* Set systick reload value to generate 1ms interrupt */
68 if (SysTick_Config(SystemCoreClock / 1000U))
69
```

– Causes HardFault Exception

- Debug and let it fault
- vectpc pseudo register shows fault details

The screenshot shows the IDE's 'Faults' view for the file 'led_blinky.c' at line 69. It displays active faults, including a Bus Fault (BFSR) and a Hard Fault (HFSR). The HFSR is 'FORCED (30)', indicating a forced hard fault. Below this, there are two tables: 'Fault Status Registers' and 'Stacked Registers (LR/EXC_RETURN=0xfffffff9)'. A red arrow points from the 'vectpc' pseudo register in the 'Stacked Registers' table to the 'main()' function in the call stack.

Active faults @ led_blinky.c [line 69]

Bus Fault (BFSR)

- IMPRECISERR (2) Imprecise data bus error

Hard Fault (HFSR)

- FORCED (30) Indicates a forced hard fault, generated by escalation of a fault with configurable priority that cannot be handled, either because of priority or because it is disabled

Fault Status Registers

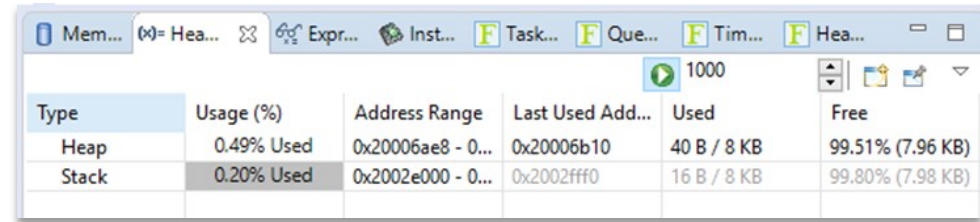
Name	Value	Description
IPSR	0x00000003	Exception Status Register (Hard Fault)
CFSR	0x00000400	Configurable fault Status Register
BFSR	0x00000004	Bus fault Status Register
HFSR	0x40000000	Hard fault Status Register
DFSR	0x00000000	Debug fault Status Register
AFSR	0x00000000	Auxiliary fault Status Register

Stacked Registers (LR/EXC_RETURN=0xfffffff9)

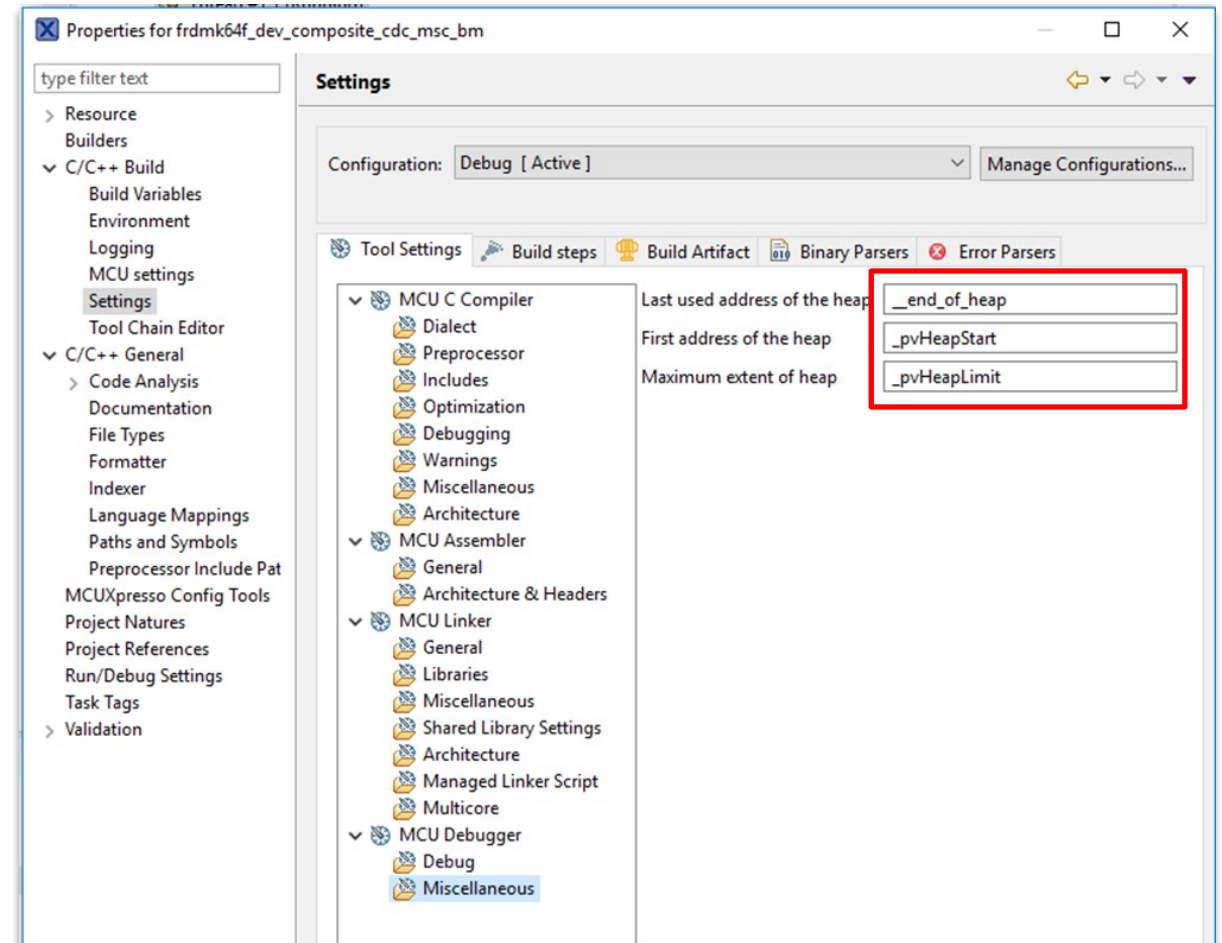
Name	Value	Description
R0	0x400FF040	
R1	0x00000001	
R2	0x00000000	
R3	0x20000008	
R12	0x00000000	
LR	0x00000E3D	↳ GPIO_PinInit()
PC	0x00000626	↳ main()
PSR	0x41000000	
SP	0x2002FFC8	

Heap and Stack views

- Real-time polling of Heap usage
- Visual indication (Red/Orange/Green) of usage vs limits
- Heap limit set in Project properties
- Stack cannot be read real time (as target would have to be stopped)



Type	Usage (%)	Address Range	Last Used Add...	Used	Free
Heap	0.49% Used	0x20006ae8 - 0...	0x20006b10	40 B / 8 KB	99.51% (7.96 KB)
Stack	0.20% Used	0x2002e000 - 0...	0x2002ffff0	16 B / 8 KB	99.80% (7.98 KB)



Properties for frdmk64f_dev_composite_cdc_msc_bm

Settings

Configuration: Debug [Active] Manage Configurations...

Tool Settings Build steps Build Artifact Binary Parsers Error Parsers

MCU C Compiler

- Dialect
- Preprocessor
- Includes
- Optimization
- Debugging
- Warnings
- Miscellaneous
- Architecture

MCU Assembler

- General
- Architecture & Headers

MCU Linker

- General
- Libraries
- Miscellaneous
- Shared Library Settings
- Architecture
- Managed Linker Script
- Multicore

MCU Debugger

- Debug
- Miscellaneous

Last used address of the heap:

First address of the heap:

Maximum extent of heap:

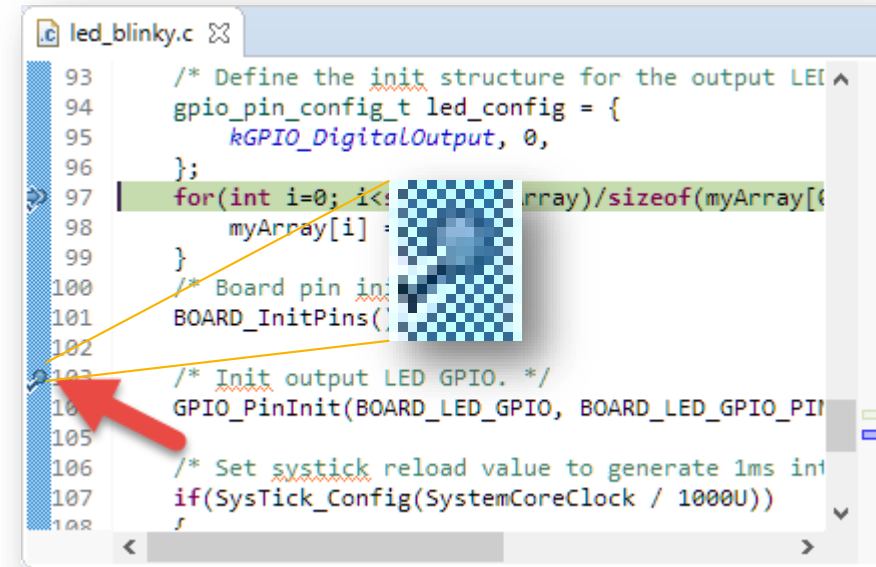
PART 3: CODE & DATA BREAKPOINTS

About this section

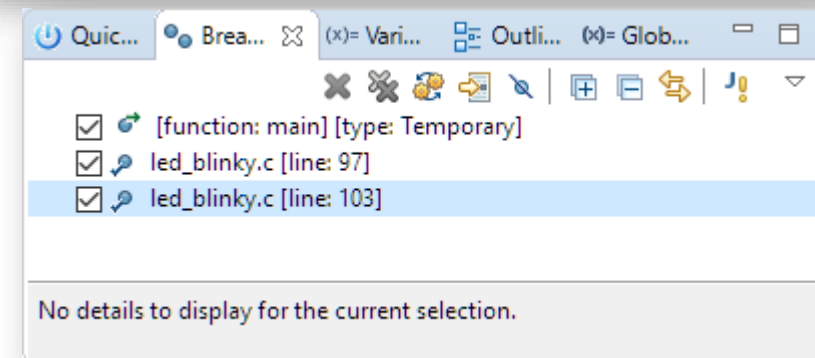
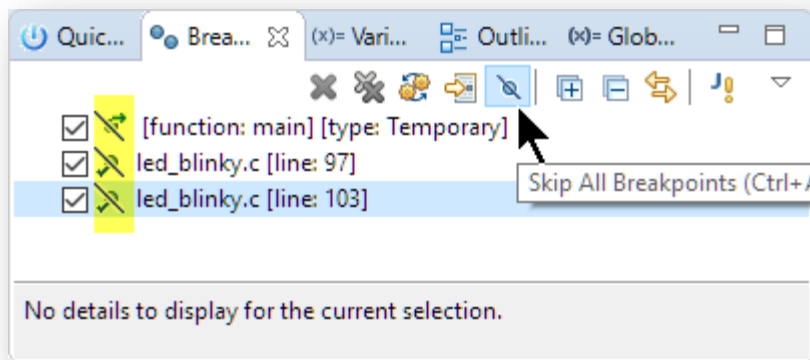
- Screen shots shown are from a session with FRDM-K64F
- All operations also apply to any MCUXpresso SDK supported board

Breakpoints and Breakpoints View

- Add Breakpoint to source view
 - Double-Click into 'blue' ribbon
- Breakpoints are listed in Breakpoints view
- Set/Enabled breakpoint have checkmark
- Be aware of 'Skip All Breakpoints'!

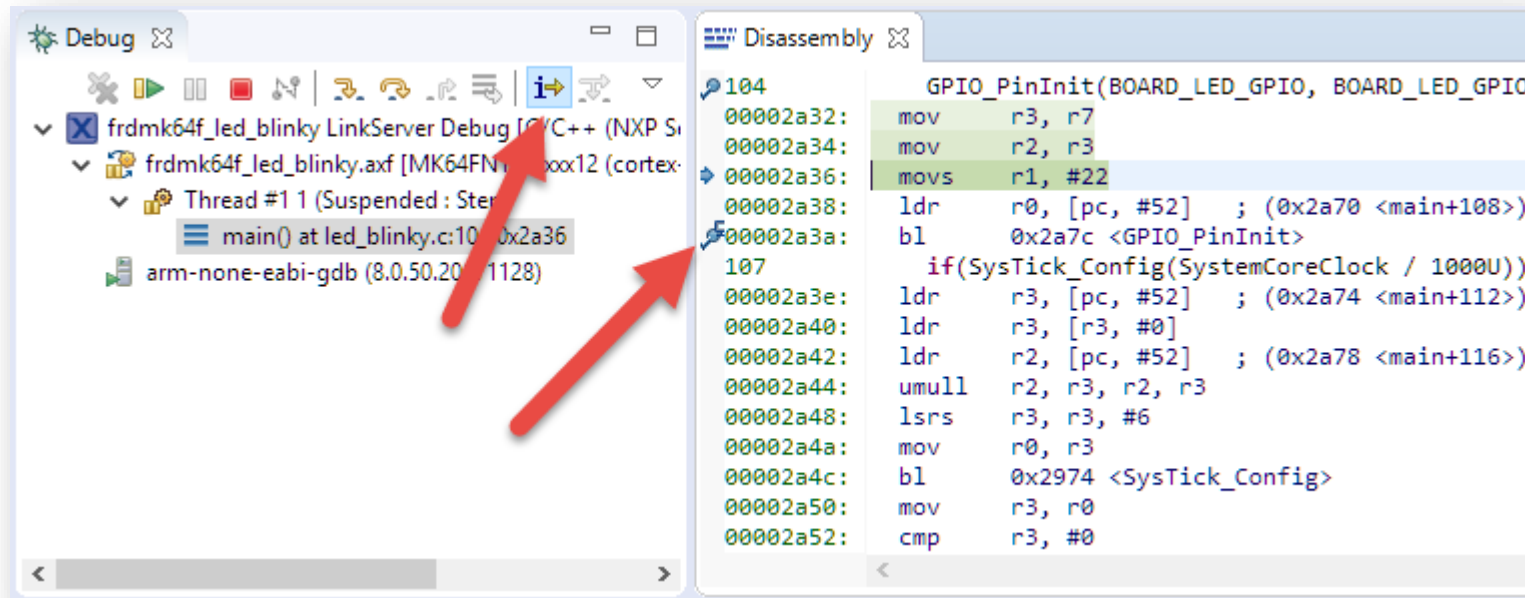


```
93  /* Define the init structure for the output LED
94  gpio_pin_config_t led_config = {
95      kGPIO_DigitalOutput, 0,
96  };
97  for(int i=0; i<sizeof(myArray)/sizeof(myArray[0]); i++)
98      myArray[i] = 0;
99  }
100 /* Board pin in
101 BOARD_InitPins()
102
103 /* Init output LED GPIO. */
104 GPIO_PinInit(BOARD_LED_GPIO, BOARD_LED_GPIO_PIN)
105
106 /* Set systick reload value to generate 1ms int
107 if(SysTick_Config(SystemCoreClock / 1000U))
108
```



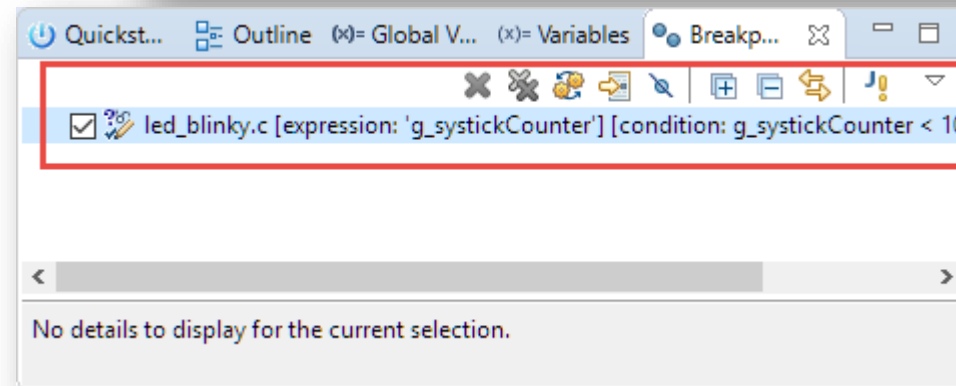
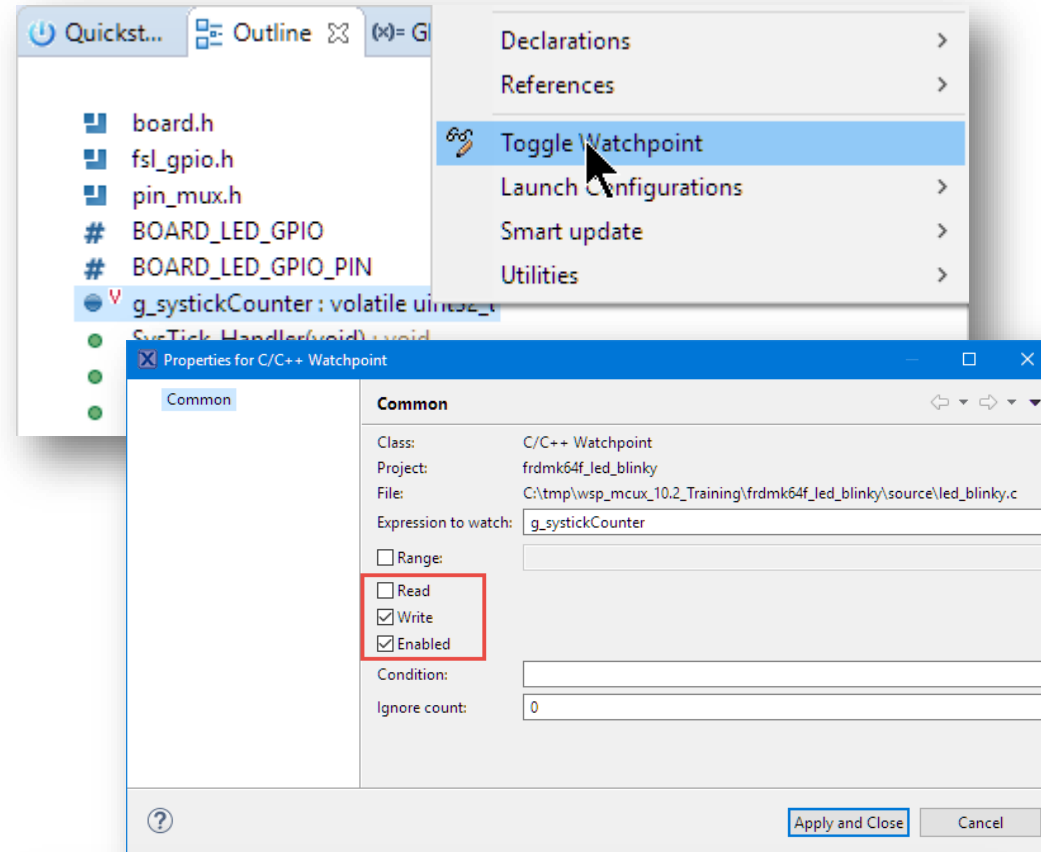
Assembly Stepping and Breakpoints

- Turn on Instruction Stepping 
- Add/Remove assembly instruction breakpoints 



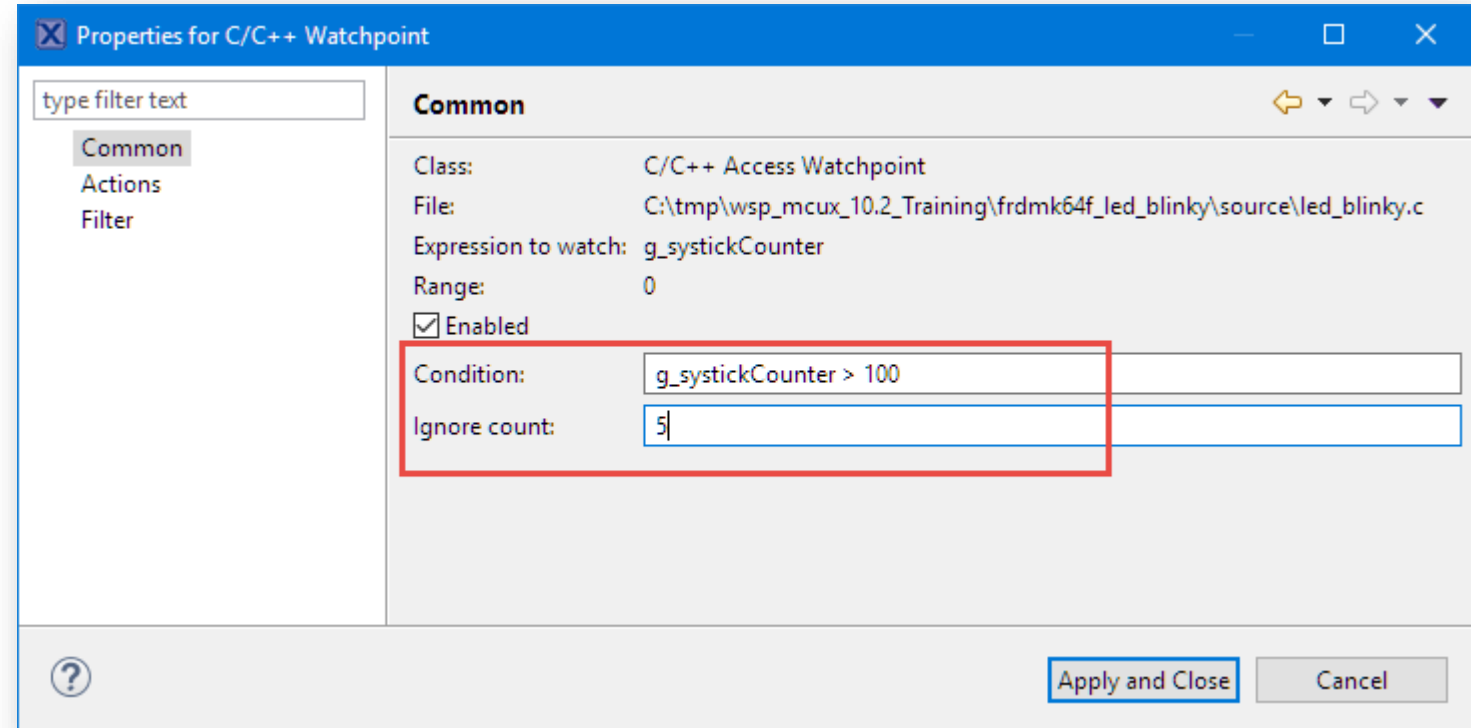
Watchpoints

- Breakpoint on data read/write
- **Debug**
- Use **Outline View**
- **Toggle Watchpoint**
- Read/**Write** trigger condition
- Apply and Close
- **Run Target** → stops on access
- Use Breakpoint View to create/update/delete watchpoints



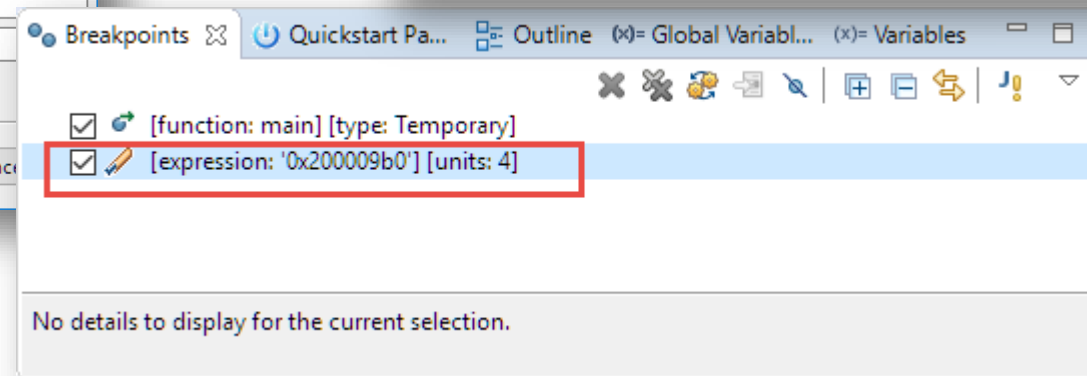
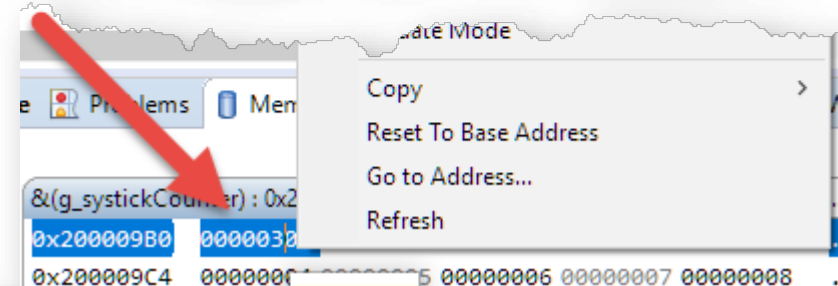
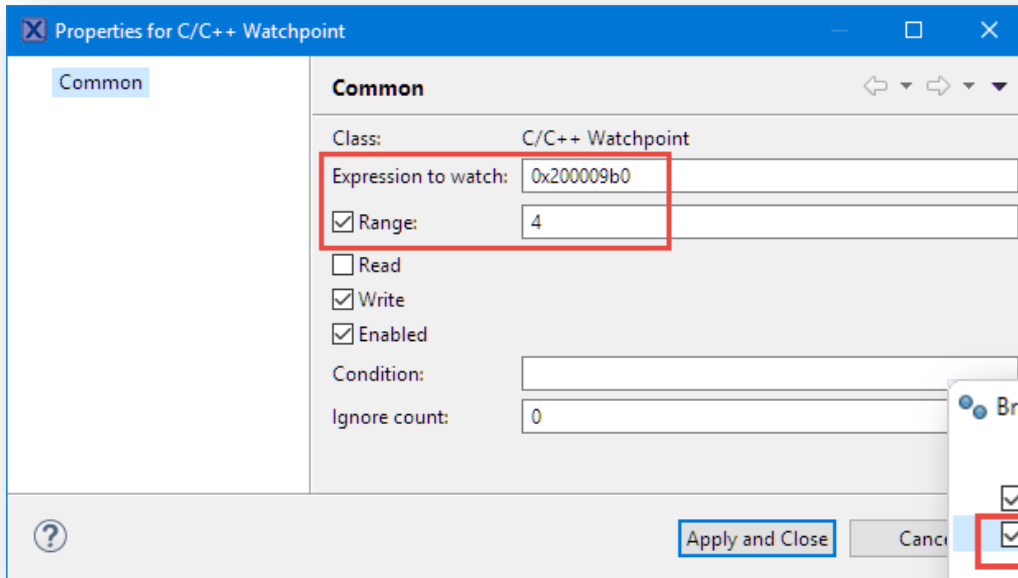
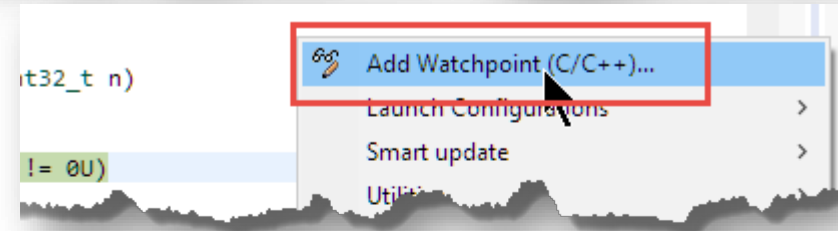
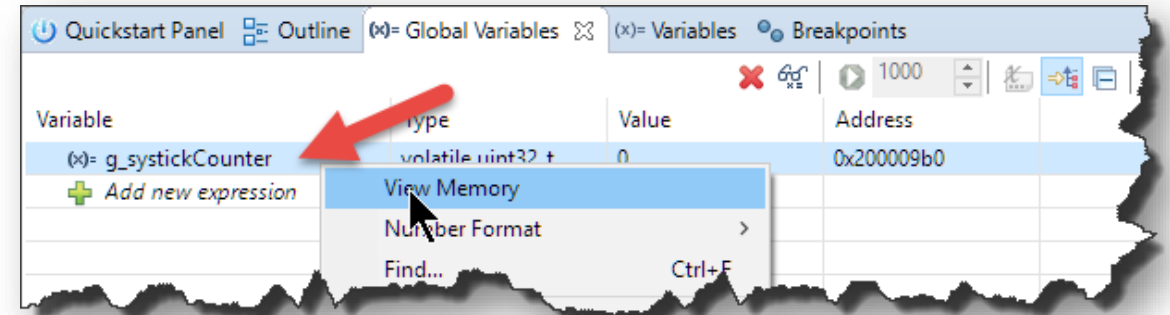
Watchpoint Properties

- **Read/Write Access** change:
 - Limitation of CDT: re-create watchpoint
- **Condition:** only stops on access if condition is true
- **Ignore Count:** counts down to zero until it stops
- **Actions & Filter**
- *Note: stops target for condition evaluation!*



Watchpoints from Memory View

- View Memory on Global Variable
- In Memory View, select range
- Add Watchpoint (C/C++)
- Watchpoint on address range



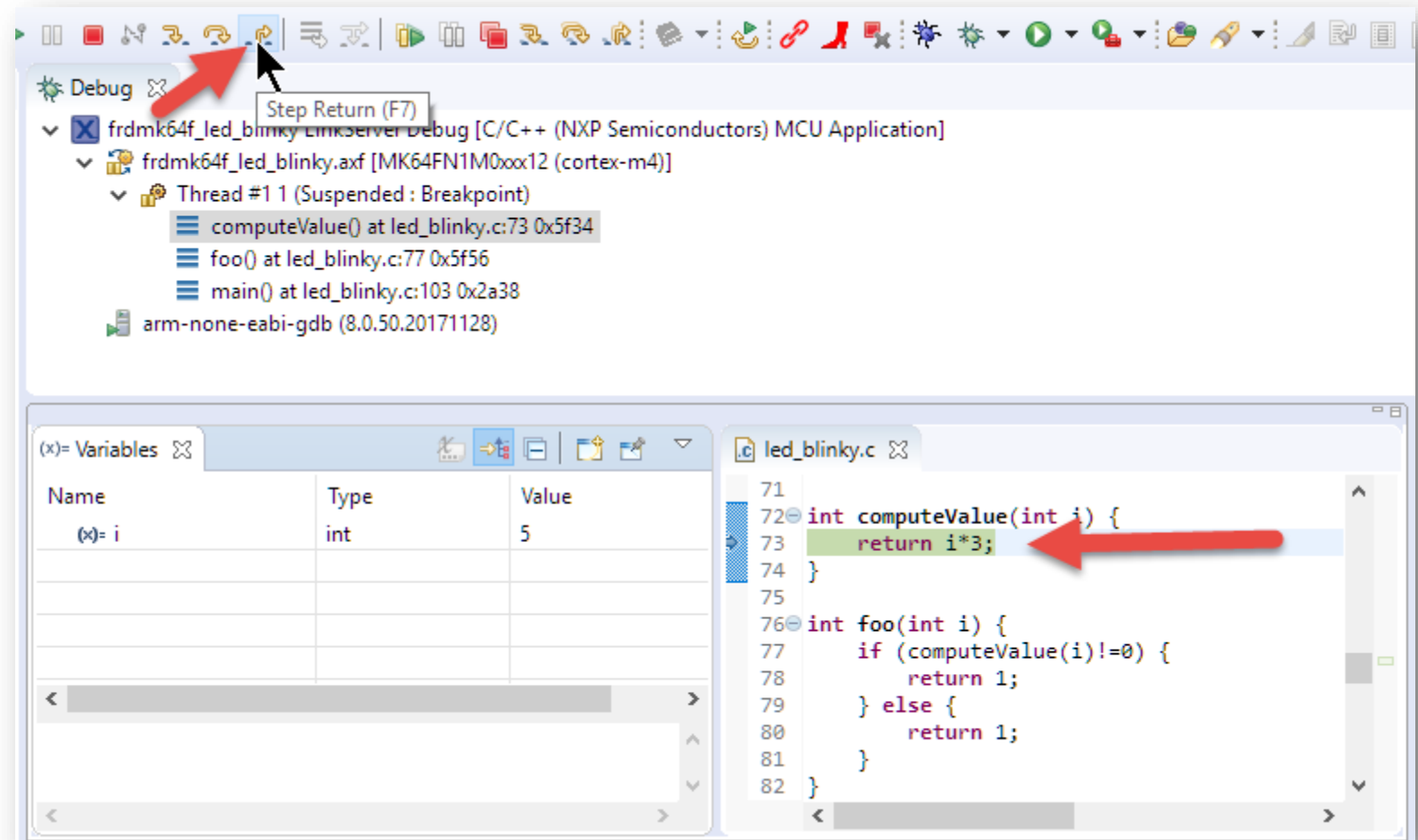
Stepping Return Value of Functions

- Problem
 - How to see the calculated return value
 - Usually calculated/returned in registers
 - No local variable to inspect
- Solution
 - Step-Return Value in Variable view
- Add code to application and build (copy-paste or use snippet)

```
int computeValue(int i) {  
    return i*3;  
}  
  
int foo(int i) {  
    if (computeValue(i)!=0) {  
        return 1;  
    } else {  
        return 1;  
    }  
}
```

Step-Return Value

- Debug
- Perform a Step-Return from Function which returns value



Step-Return Value

- After the Step-Return, return value is shown in Variables View

The screenshot shows a debugger interface with three main panels. The top panel, titled 'Debug', shows the call stack with 'foo() at led_blinky.c:77 0x5f56' selected. The bottom-left panel, titled '(x)= Variables', displays a table of variables. The variable '(x)= \$2' is highlighted with a red box, showing a value of 15. The bottom-right panel shows the source code for 'led_blinky.c', with the 'if' statement on line 77 highlighted in green, indicating the current execution point.

Name	Type	Value
(x)= \$2	int	15
(x)= i	int	5

```
69     }
70 }
71
72 int computeValue(int i) {
73     return i*3;
74 }
75
76 int foo(int i) {
77     if (computeValue(i)!=0) {
78         return 1;
79     } else {
80         return 1;
81     }
82 }
```

PART 4: INSTRUCTION TRACE

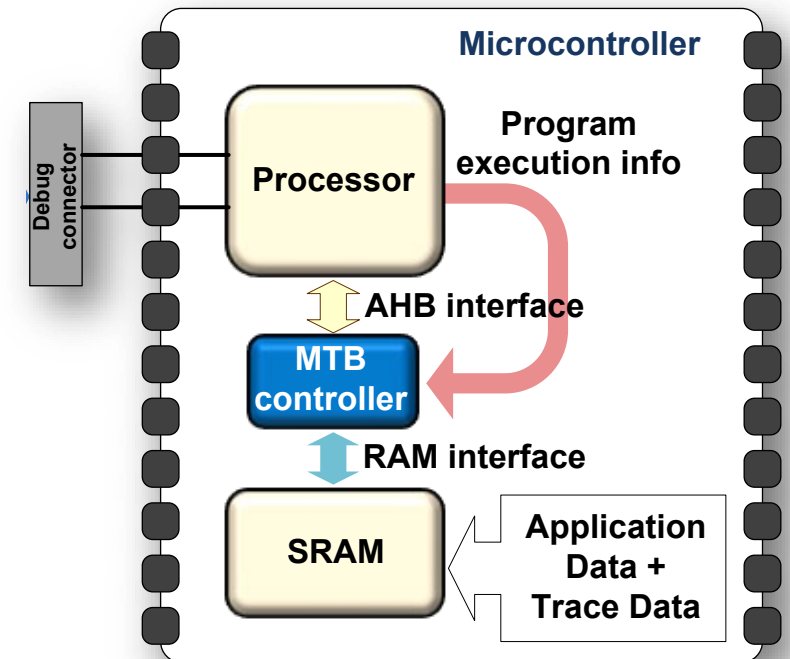


About this section

- Screen shots shown are from a session with FRDM-K64F (for ETM/ETB) and an LPC845 Breakout (BRK) board (for MTB)
- Instruction trace availability shown on the next slide

Instruction Trace – Supported Targets in MCUXpresso IDE

- Cortex M3/M4/M7 MCUs
 - Target MCU must implement both an Embedded Trace Macrocell (ETM) AND an Embedded Trace Buffer (ETB).
 - ✓ Kinetis K, LPC18xx and LPC43xx parts
- Cortex M0+ LPC/Kinetis MCUs
 - Target MCU must implement a Micro Trace Buffer (MTB)
 - ✓ LPC81x/82x/84x parts
 - ✓ LPC11U6x/11E6x parts
 - ✓ Kinetis L parts
- For parts with ETM only (LPC546xx/0xx, i.MX RT), recommend debug probe and trace solutions from partners
 - E.g. SEGGER, PE Micro, Arm Keil, IAR, Percepio, Lauterbach, iSYSTEM



Instruction Trace in MCUXpresso can be carried out via any supported debug probe

Instruction Trace

- Collects details of instructions being executed
- Allows complex program flow problems to be examined
 - Gives insight into what happening in system before a fault was encountered
- [<IDE Installation Path>\MCUXpresso_IDE_Instruction_Trace.pdf](#)

The screenshot displays the MCUXpresso IDE interface with three main views:

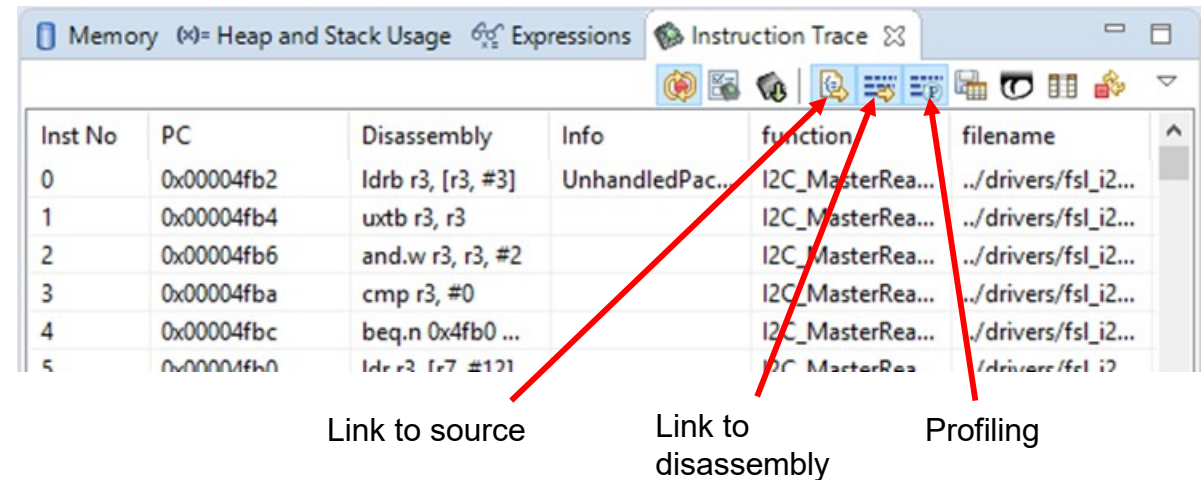
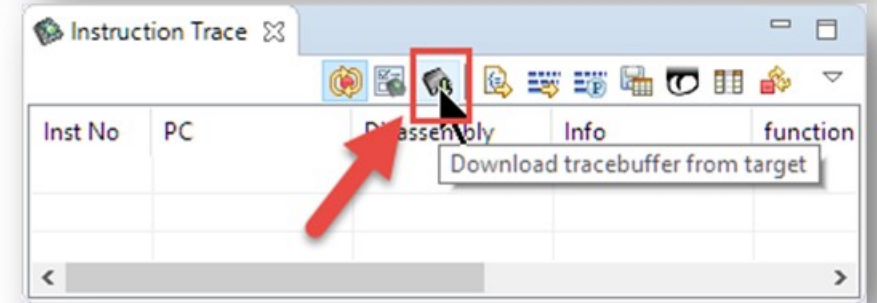
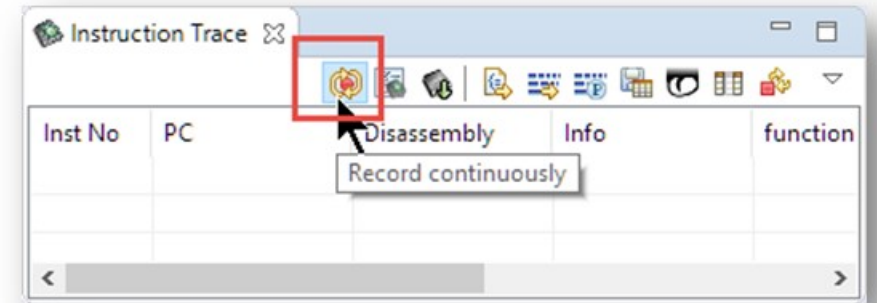
- Code coverage:** The top-left pane shows C source code with green and blue highlights. Annotations include "Code coverage" pointing to the highlighted lines and "Current PC" pointing to the current execution line (line 105).
- Disassembly:** The top-right pane shows the corresponding assembly instructions. An annotation "Code coverage counters" points to the instruction list.
- Instruction Trace View:** The bottom pane is a table listing executed instructions. An annotation "Can be linked to src / asm views" points to the table.

Inst No	PC	Disassembly	Info	function	filename	line no
2344	0x00000408	lsls r3, r3, #2		main	./src/main.c	98
2345	0x0000040a	cmp r2, r3		main	./src/main.c	98
2346	0x0000040c	bls.n 0x426 <...>		main	./src/main.c	98
2347	0x0000040e	ldr r3, [pc, #56...		main	./src/main.c	98
2348	0x00000410	ldr r2, [r3, #0]		main	./src/main.c	98
2349	0x00000412	movs r3, #150 ...		main	./src/main.c	98
2350	0x00000414	lsls r3, r3, #3		main	./src/main.c	98
2351	0x00000416	cmp r2, r3		main	./src/main.c	98
2352	0x00000418	bhi.n 0x426 <...>		main	./src/main.c	98







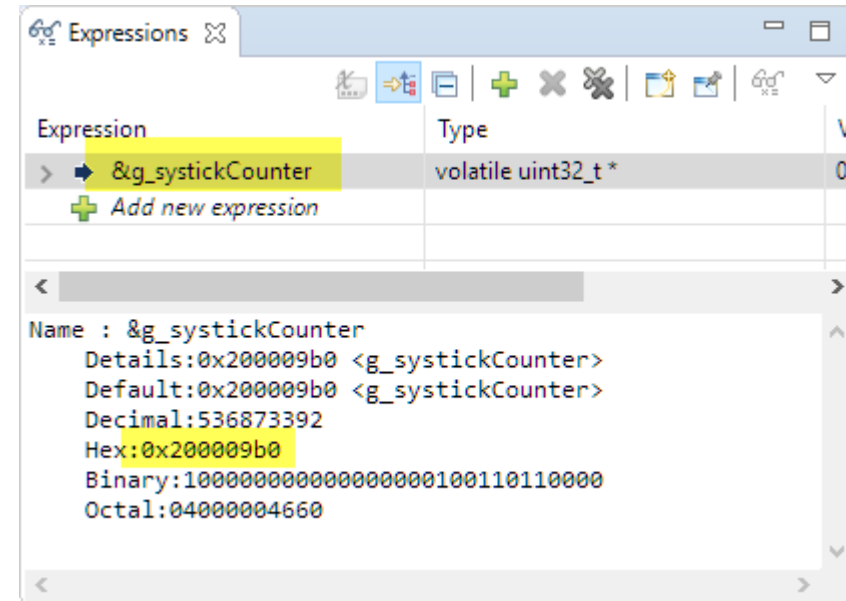
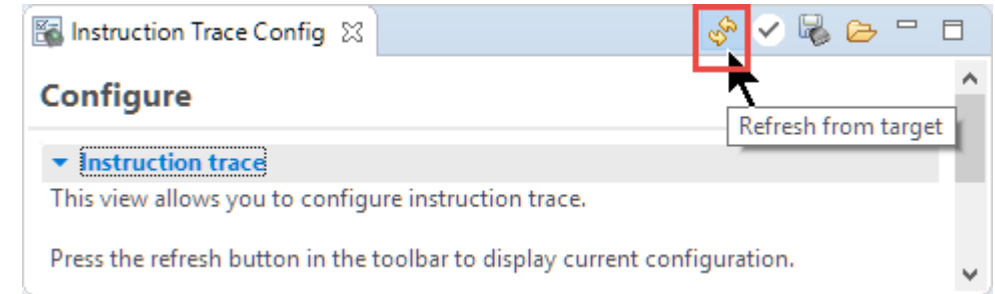
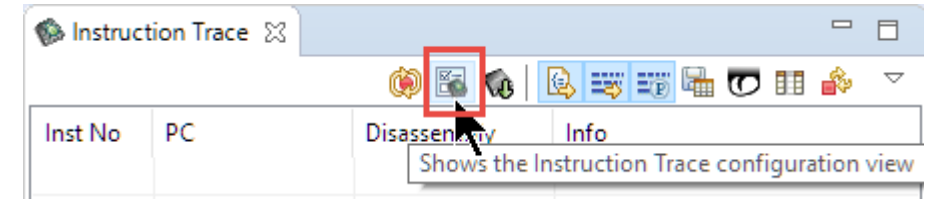
Instruction Trace

- **Debug Application**
 - bubble demo application better than led_blinky
- **Instruction Trace View**
 - If not visible, access by typing "Instruction trace in Quick Access box (top right)
 - **Turn on 'Record continuously' Button**
- **Run the Target and then Suspend/Pause**
- **Download tracebuffer from target**
- Turn on **Link to Source & Disassembly**
- Shows corresponding program location




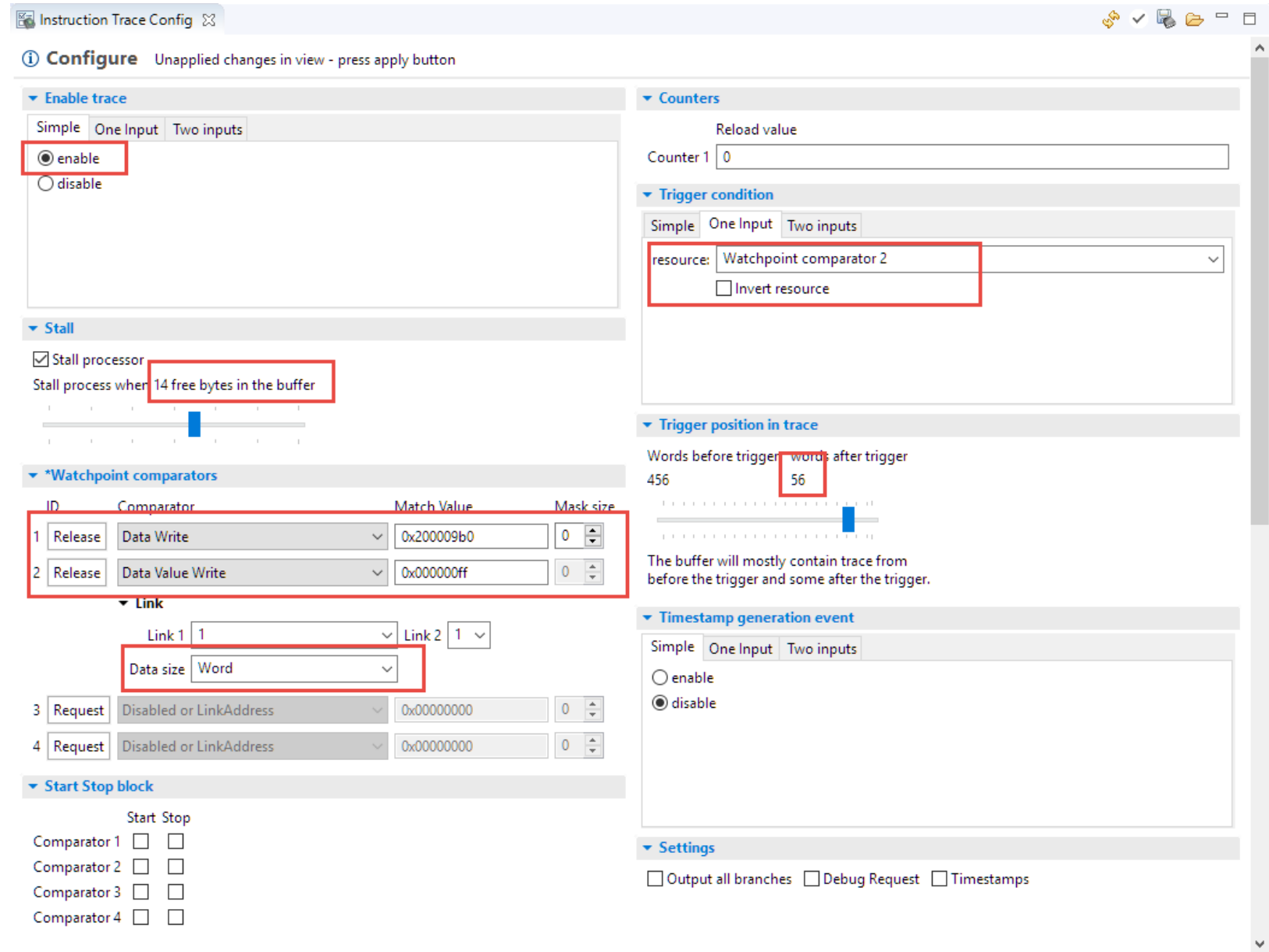
Instruction Trace Triggers

- Debug and Suspend 
- Show Instruction Trace Configuration 
- Refresh from target 
- Address of global variable
 - Expressions View 
 - Add Expression
 - Address: &<variableName>
 - Copy-Paste address



Instruction Trace Trigger Configuration

- Trace enabled
- Stall if only some free bytes in buffer
- Data write on address and value written
- Trigger on comparator 2
- Records some words before and after trigger
- Save Configuration 



The screenshot shows the 'Instruction Trace Config' window with the following settings:

- Enable trace:** enable
- Stall:** Stall processor. Stall process when: 14 free bytes in the buffer.
- *Watchpoint comparators:**

ID	Comparator	Match Value	Mask size
1	Release Data Write	0x200009b0	0
2	Release Data Value Write	0x000000ff	0

Link: Link 1: 1, Link 2: 1. Data size: Word.
- Counters:** Reload value: 0. Counter 1: 0.
- Trigger condition:** resource: Watchpoint comparator 2. Invert resource.
- Trigger position in trace:** Words before trigger: 456, words after trigger: 56.
- Timestamp generation event:** disable.
- Settings:** Output all branches, Debug Request, Timestamps.

PART 5: FREERTOS TASK AWARE DEBUG

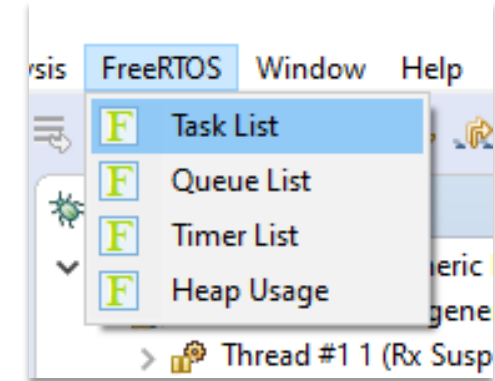


About this section

- Screen shots shown are from a session with FRDM-K64F
- All operations also apply to any MCUXpresso SDK supported board that includes FreeRTOS support

FreeRTOS TAD

- Thread Aware Debugging
 - Show and switch between threads in Debug View
 - Views to inspect status of the RTOS
- Views read RTOS data structures while target is halted
- Debugger needs extra information:
 - [<IDE Installation Path>\MCUXpresso_IDE_FreeRTOS_Debug_Guide.pdf](#)
 - SDK projects should be updated for this



Task List (FreeRTOS)

TCB#	Task Name	Task Handle	Task State	Prior...	Stack Usage	Ever
> 1	Rx	0x20000520	Suspended	2 (2)	336 B / 1016 B	Unk
> 2	TX	0x200009a8	Blocked	1 (1)	160 B / 1016 B	
> 3	Sem	0x20000e30	Suspended	4 (4)	300 B / 1016 B	Unk
> 4	IDLE	0x20001150	Running	0 (0)		
> 5	Tmr Svc	0x200014a8	Blocked	4 (4)		

Heap Usage (FreeRTOS)

Type	Heap Base	Heap End	Heap Usage	Free Space	Heap Usage Graph
4	0x20000054	0x20002854	5.21 kB / 10 kB	47.89% (4.79 kB)	52.11% Used

Queue List (FreeRTOS)

#	Queue Name	Address	Length	Item Size	# Tx Wai...	# Rx Wai...	Queue Type
1	TmrQ	0x20000ee8	0/10	0xffffffff (-1 B)	0	1	Queue

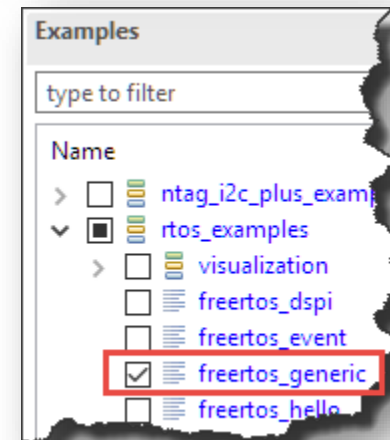


Creating FreeRTOS Project

- Quickstart Panel
- Import SDK example(s)
- Select Board (FRDM-K64F)
- `rtos_example > freertos_generic`
- Finish to create project
- Build project

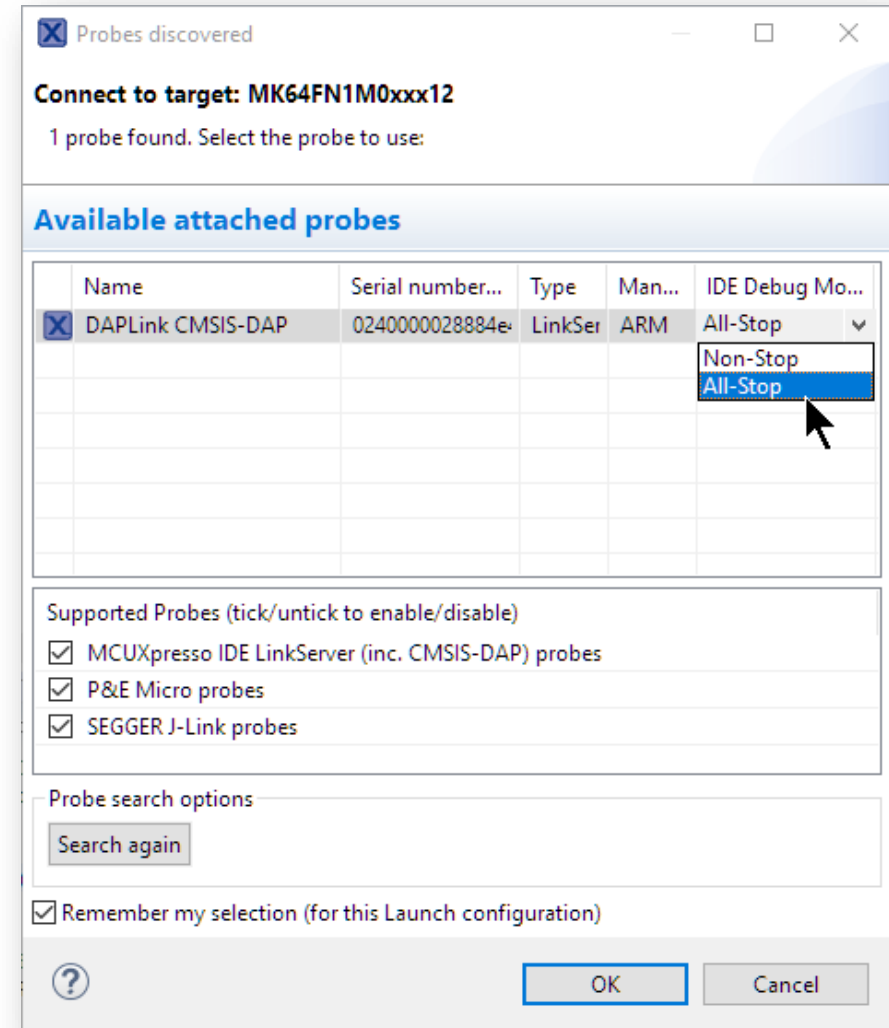
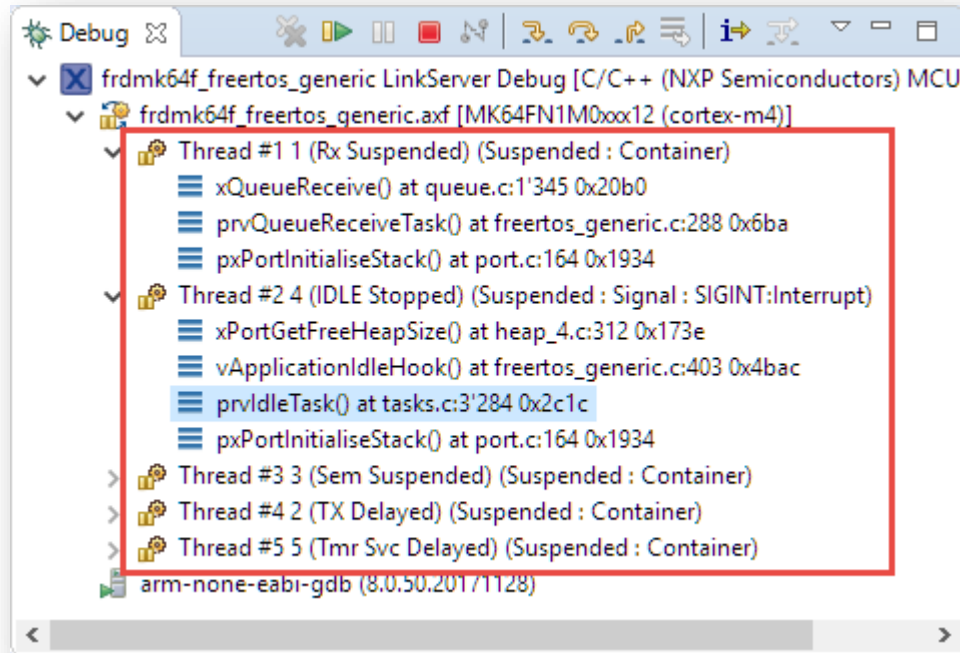
▼ Create or import a project

- New project...
- Import SDK example(s)...
- Import project(s) from file system...



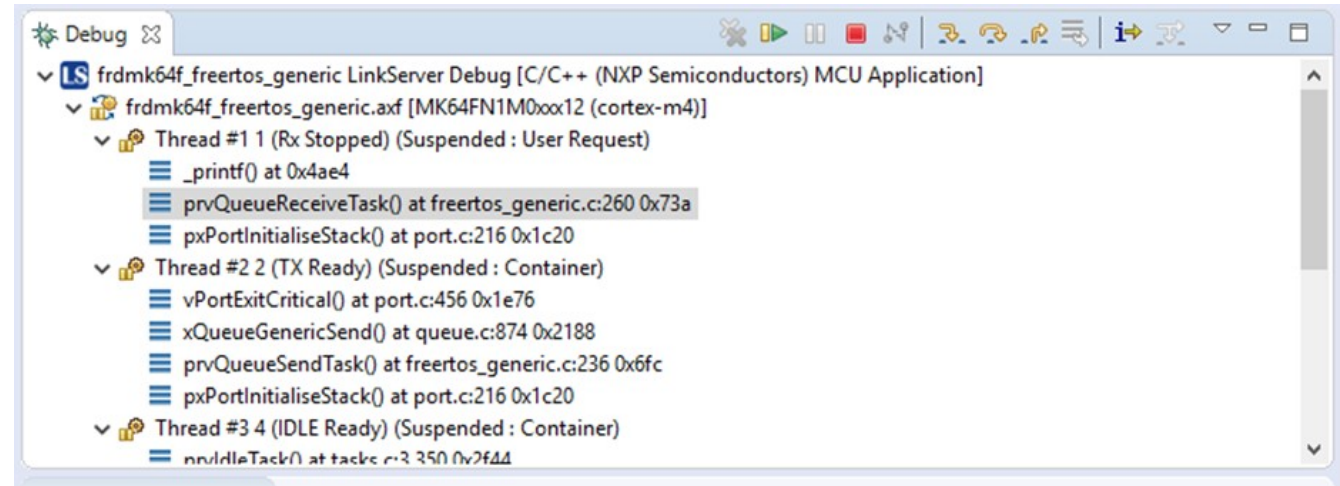
Debug Probe Connection

- Configure to use **All-Stop**
 - Allows thread aware debug view
 - Otherwise only current thread is shown
- **Resume**, then **Pause**



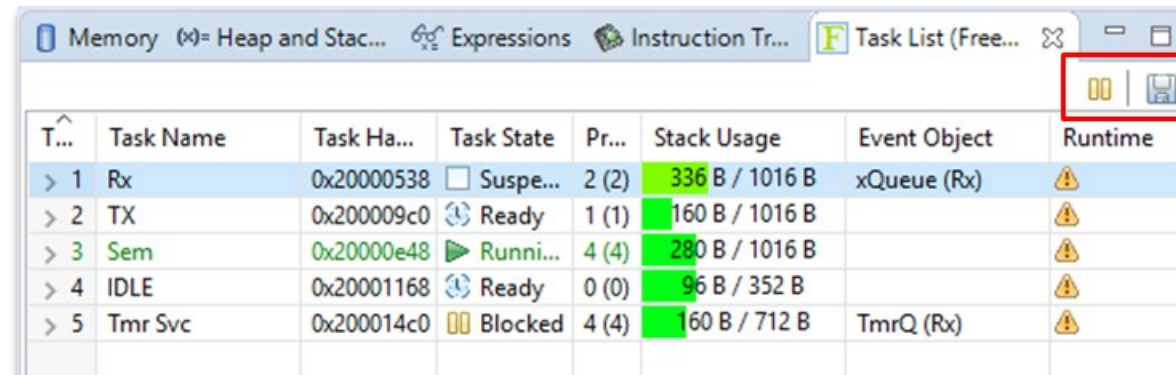
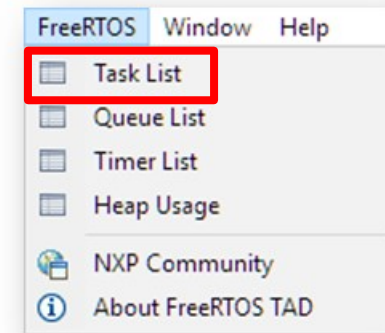
Thread Aware Debug View

- Lists all FreeRTOS **tasks** with stacks
 - Switch debug context to thread
 - Context, registers, stack
- **Click on function** (top) of a thread
- Debug it (**step out, step over**)
- **Switch** to another thread



Task List

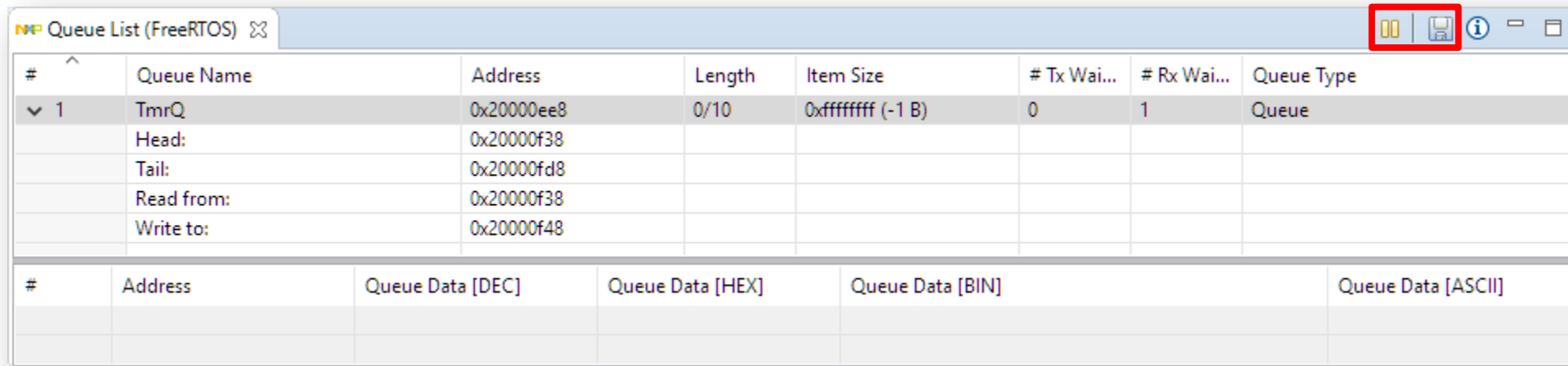
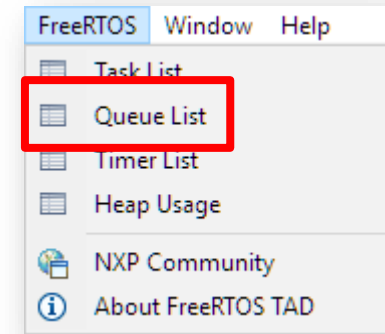
- Lists FreeRTOS Tasks in the System
- Menu **FreeRTOS > Task List**
- **Pause** Button: do not update data when target is stopped
- **Save** Button: Store information as .csv file

A screenshot of the 'Task List (Free...)' window in the software. The window title bar includes 'Memory', 'Heap and Stac...', 'Expressions', 'Instruction Tr...', and 'Task List (Free...'. In the top right corner of the window, there are two icons: a pause icon and a save icon, both highlighted with a red rectangular box. Below the icons is a table with the following data:

T...	Task Name	Task Ha...	Task State	Pr...	Stack Usage	Event Object	Runtime
> 1	Rx	0x20000538	<input type="checkbox"/> Suspe...	2 (2)	336 B / 1016 B	xQueue (Rx)	⚠
> 2	TX	0x200009c0	<input checked="" type="checkbox"/> Ready	1 (1)	160 B / 1016 B		⚠
> 3	Sem	0x20000e48	<input checked="" type="checkbox"/> Runni...	4 (4)	280 B / 1016 B		⚠
> 4	IDLE	0x20001168	<input checked="" type="checkbox"/> Ready	0 (0)	96 B / 352 B		⚠
> 5	Tmr Svc	0x200014c0	<input checked="" type="checkbox"/> Blocked	4 (4)	160 B / 712 B	TmrQ (Rx)	⚠

Queue List

- Lists all FreeRTOS **Queues**
- Menu **FreeRTOS > Queue List**
- **Pause** Button: do not update data when target is stopped
- **Save** Button: Store information as .csv file

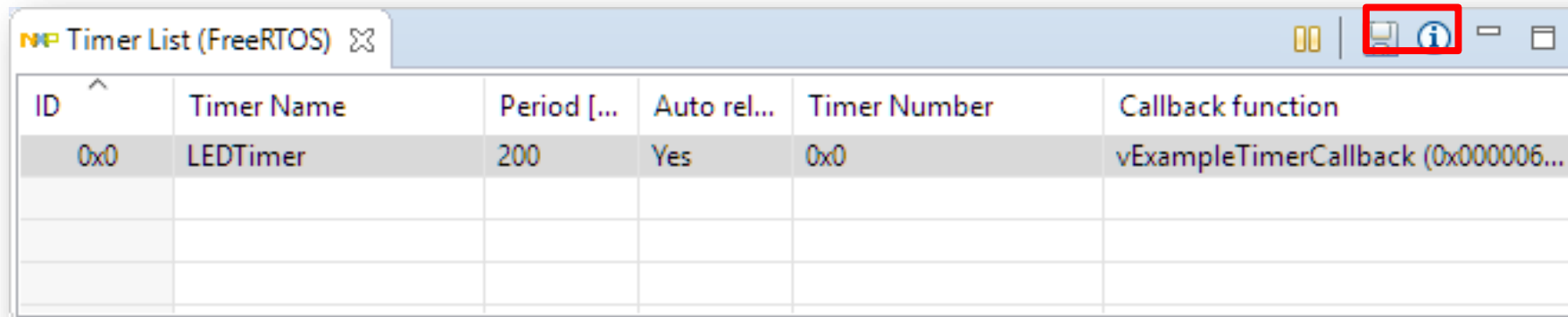
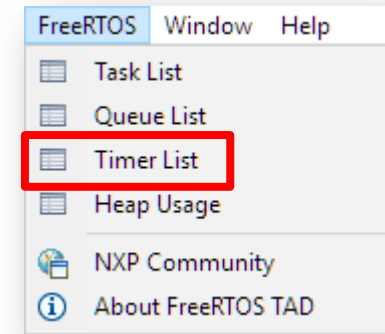


#	Queue Name	Address	Length	Item Size	# Tx Wai...	# Rx Wai...	Queue Type
1	TmrQ	0x20000ee8	0/10	0xffffffff (-1 B)	0	1	Queue
	Head:	0x20000f38					
	Tail:	0x20000fd8					
	Read from:	0x20000f38					
	Write to:	0x20000f48					

#	Address	Queue Data [DEC]	Queue Data [HEX]	Queue Data [BIN]	Queue Data [ASCII]

Queue Timer

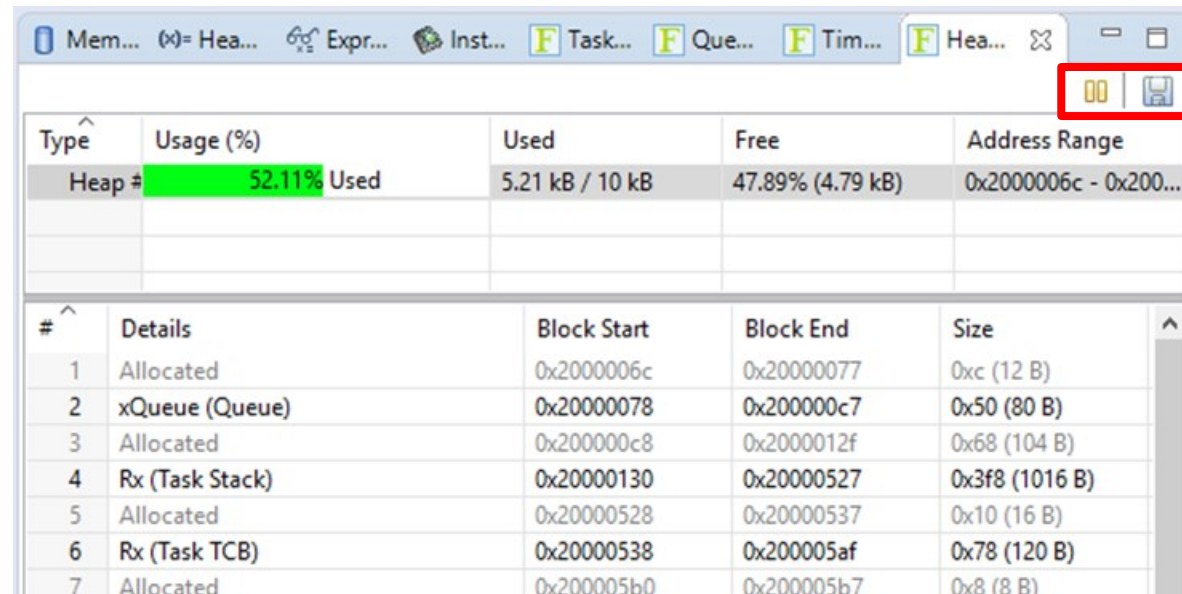
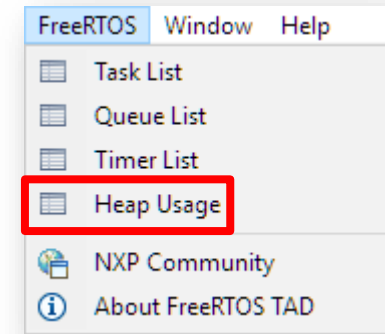
- Lists all FreeRTOS **Software Timer**
- Menu **FreeRTOS > Timer List**
- **Pause** Button: do not update data when target is stopped
- **Save** Button: Store information as .csv file

A screenshot of the 'Timer List (FreeRTOS)' window. The window title bar includes a 'Save' icon (floppy disk) and an 'Info' icon (i), both of which are highlighted with a red box. The main content is a table with the following data:

ID ^	Timer Name	Period [...]	Auto rel...	Timer Number	Callback function
0x0	LEDTimer	200	Yes	0x0	vExampleTimerCallback (0x000006...

Heap Usage

- Status of FreeRTOS **Heap** and **Memory Allocation**
- Menu **FreeRTOS > Queue List**
- **Pause** Button: do not update data when target is stopped
- **Save** Button: Store information as .csv file



The screenshot shows the 'Heap Usage' window in the FreeRTOS IDE. The window title bar includes 'Mem...', '(*)= Hea...', 'Expr...', 'Inst...', 'Task...', 'Que...', 'Tim...', and 'Hea...'. The main content area displays a table with the following data:

Type	Usage (%)	Used	Free	Address Range
Heap #	52.11% Used	5.21 kB / 10 kB	47.89% (4.79 kB)	0x2000006c - 0x200...

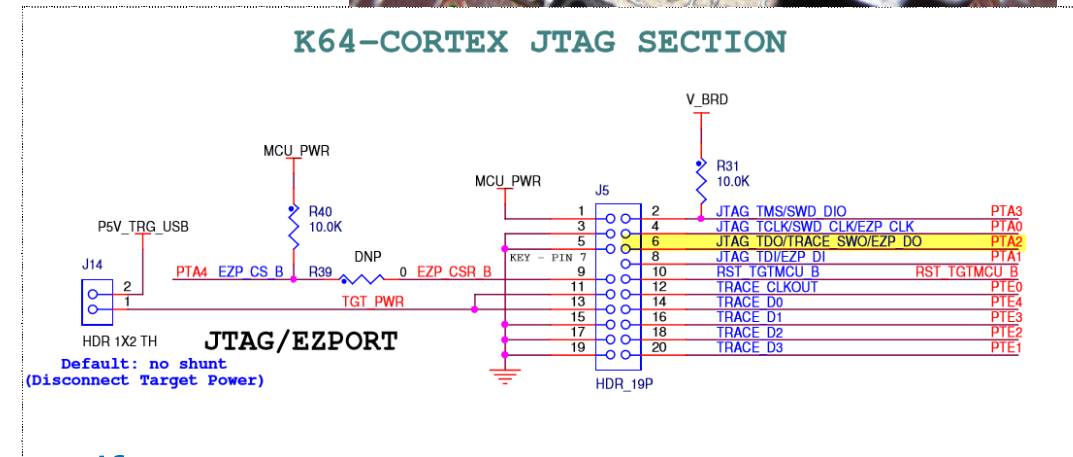
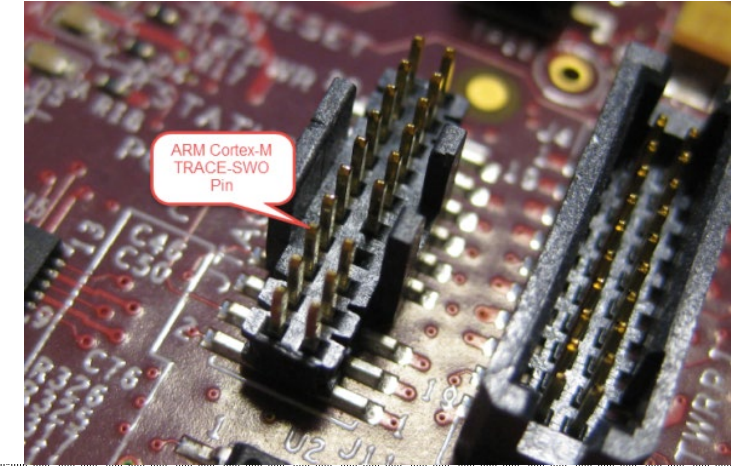
#	Details	Block Start	Block End	Size
1	Allocated	0x2000006c	0x20000077	0xc (12 B)
2	xQueue (Queue)	0x20000078	0x200000c7	0x50 (80 B)
3	Allocated	0x200000c8	0x2000012f	0x68 (104 B)
4	Rx (Task Stack)	0x20000130	0x20000527	0x3f8 (1016 B)
5	Allocated	0x20000528	0x20000537	0x10 (16 B)
6	Rx (Task TCB)	0x20000538	0x200005af	0x78 (120 B)
7	Allocated	0x200005b0	0x200005b7	0x8 (8 B)

PART 6: SWO TRACE



Single Wire Output

- SWO: Single Wire Output
- ARM Cortex-M Hardware Feature (1 pin)
- Requires that the SWO pin is routed to debug headers
 - Present on Tower (TWR) and LPCXpresso V2/V3 boards
 - Freedom (FRDM) boards do not have SWO pin available
- Debug Features with SWO
 - SWO Console
 - SWO Interrupt tracing/logging
 - SWO Profiling
 - via LPC-Link2 with NXP CMSIS-DAP firmware
 - Using SEGGER J-Link and PE Micro probes
- [Documentation](#)
 - [<IDE Installation Path>MCUXpresso_IDE_SWO_Trace.pdf](#)

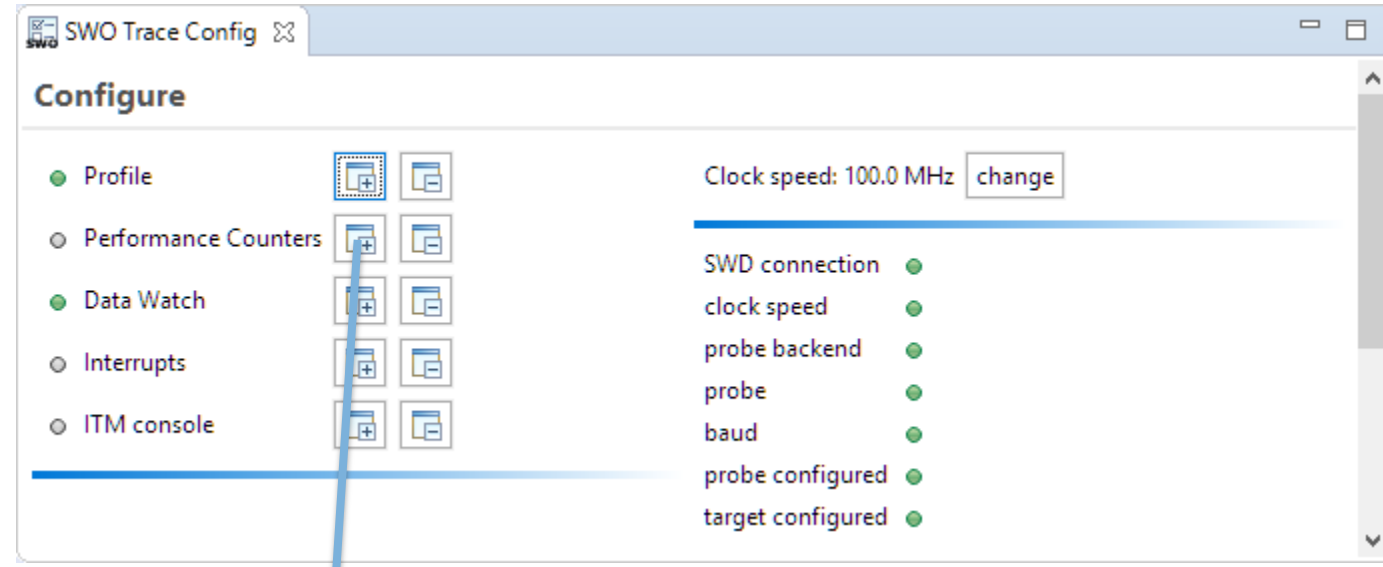


Enabling SWO use

- To use SWO, the SWO function must be enabled on the device, and clock set up to associated logic
- MCUXpresso SDK examples usually do not have this by default, but setup via pin and clock configuration tools is typically quite simple

SWO Trace Config

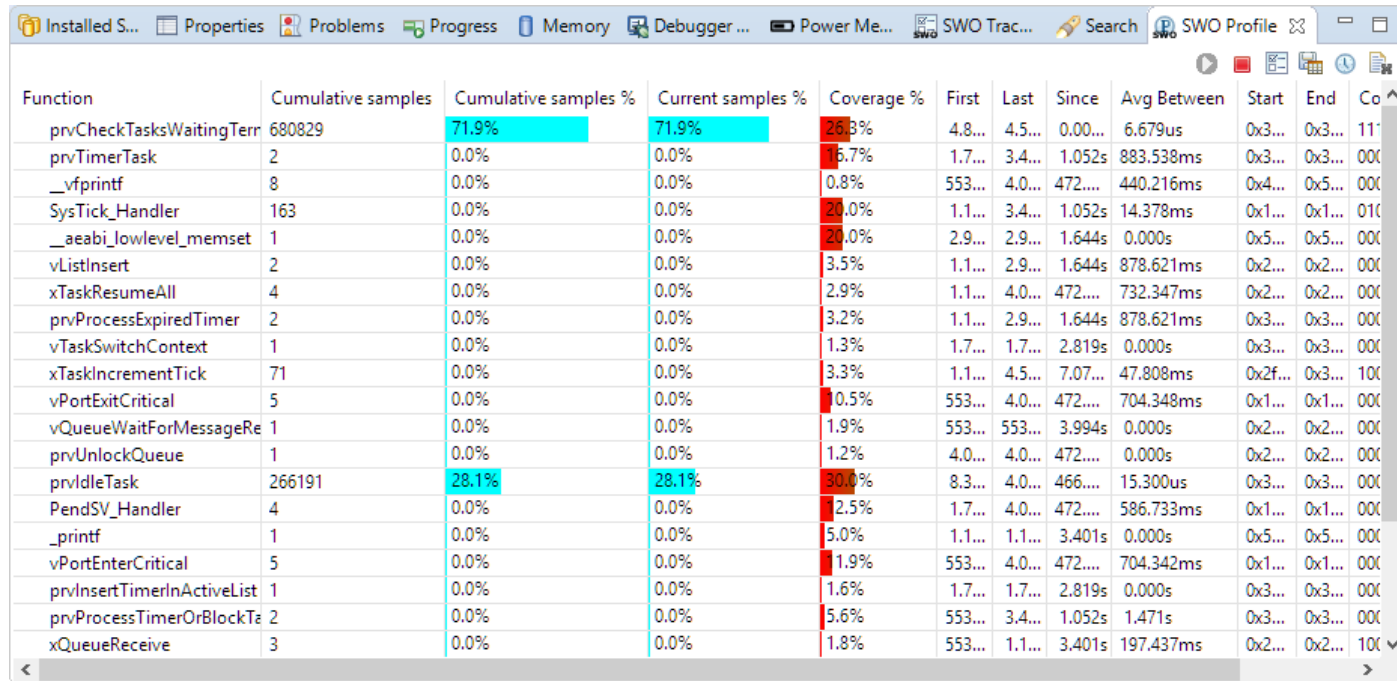
- Dedicated view to configure SWO
- SWO Clock speed setting
- (+) opens other views



Name	Total	Overflow	Counter
<input checked="" type="checkbox"/> Folded-instruction counter	0	0	0
<input checked="" type="checkbox"/> Load-store count	133	0	133
<input checked="" type="checkbox"/> Sleep overhead counter	0	0	0
<input checked="" type="checkbox"/> Exception overhead counter	0	0	0
<input checked="" type="checkbox"/> CPI	44	0	44
<input checked="" type="checkbox"/> Cycle counter	35691962	0	35691962

SWO Trace – Profile View

- Displays statistical profile of application activity
 - Based on PC sampling, typically at ~50kHz
 - Non-intrusive – does not affect application
- Benefits – Identify hotspots

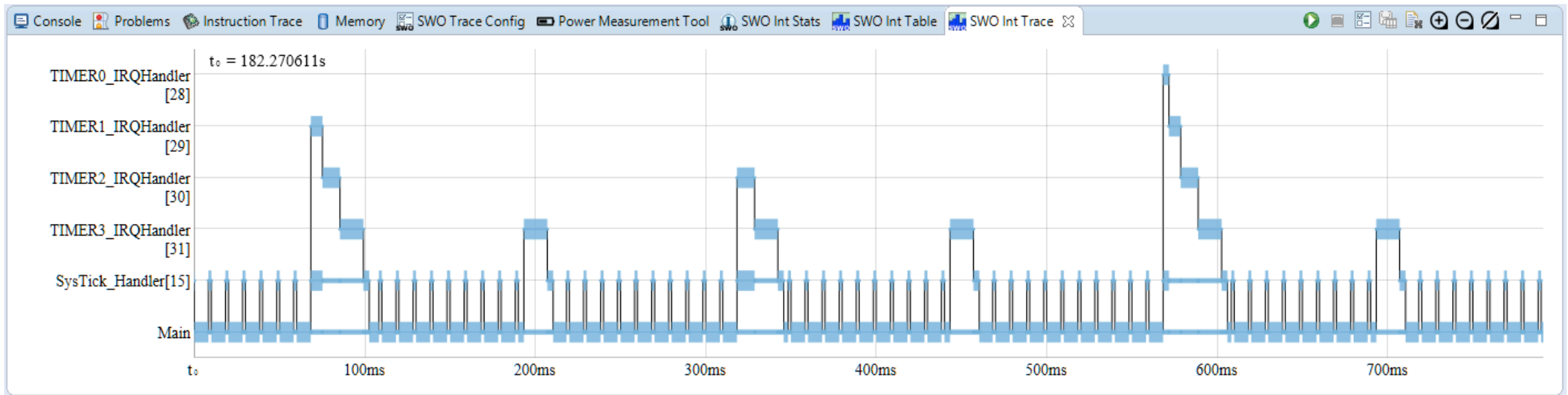


The screenshot shows the SWO Profile window with a table of function statistics. The table has columns for Function, Cumulative samples, Cumulative samples %, Current samples %, Coverage %, First, Last, Since, Avg Between, Start, End, and Co. The data is as follows:

Function	Cumulative samples	Cumulative samples %	Current samples %	Coverage %	First	Last	Since	Avg Between	Start	End	Co
prvCheckTasksWaitingTerr	680829	71.9%	71.9%	26.3%	4.8...	4.5...	0.00...	6.679us	0x3...	0x3...	11'
prvTimerTask	2	0.0%	0.0%	16.7%	1.7...	3.4...	1.052s	883.538ms	0x3...	0x3...	000
_vfprintf	8	0.0%	0.0%	0.8%	553...	4.0...	472....	440.216ms	0x4...	0x5...	000
SysTick_Handler	163	0.0%	0.0%	20.0%	1.1...	3.4...	1.052s	14.378ms	0x1...	0x1...	010
__aeabi_lowlevel_memset	1	0.0%	0.0%	20.0%	2.9...	2.9...	1.644s	0.000s	0x5...	0x5...	000
vListInsert	2	0.0%	0.0%	3.5%	1.1...	2.9...	1.644s	878.621ms	0x2...	0x2...	000
xTaskResumeAll	4	0.0%	0.0%	2.9%	1.1...	4.0...	472....	732.347ms	0x2...	0x2...	000
prvProcessExpiredTimer	2	0.0%	0.0%	3.2%	1.1...	2.9...	1.644s	878.621ms	0x3...	0x3...	000
vTaskSwitchContext	1	0.0%	0.0%	1.3%	1.7...	1.7...	2.819s	0.000s	0x3...	0x3...	000
xTaskIncrementTick	71	0.0%	0.0%	3.3%	1.1...	4.5...	7.07...	47.808ms	0x2f...	0x3...	100
vPortExitCritical	5	0.0%	0.0%	0.5%	553...	4.0...	472....	704.348ms	0x1...	0x1...	000
vQueueWaitForMessageRe	1	0.0%	0.0%	1.9%	553...	553...	3.994s	0.000s	0x2...	0x2...	000
prvUnlockQueue	1	0.0%	0.0%	1.2%	4.0...	4.0...	472....	0.000s	0x2...	0x2...	000
prvIdleTask	266191	28.1%	28.1%	30.0%	8.3...	4.0...	466....	15.300us	0x3...	0x3...	000
PendSV_Handler	4	0.0%	0.0%	2.5%	1.7...	4.0...	472....	586.733ms	0x1...	0x1...	000
_printf	1	0.0%	0.0%	5.0%	1.1...	1.1...	3.401s	0.000s	0x5...	0x5...	000
vPortEnterCritical	5	0.0%	0.0%	1.9%	553...	4.0...	472....	704.342ms	0x1...	0x1...	000
prvInsertTimerInActiveList	1	0.0%	0.0%	1.6%	1.7...	1.7...	2.819s	0.000s	0x3...	0x3...	000
prvProcessTimerOrBlockT	2	0.0%	0.0%	5.6%	553...	3.4...	1.052s	1.471s	0x3...	0x3...	000
xQueueReceive	3	0.0%	0.0%	1.8%	553...	1.1...	3.401s	197.437ms	0x2...	0x2...	100

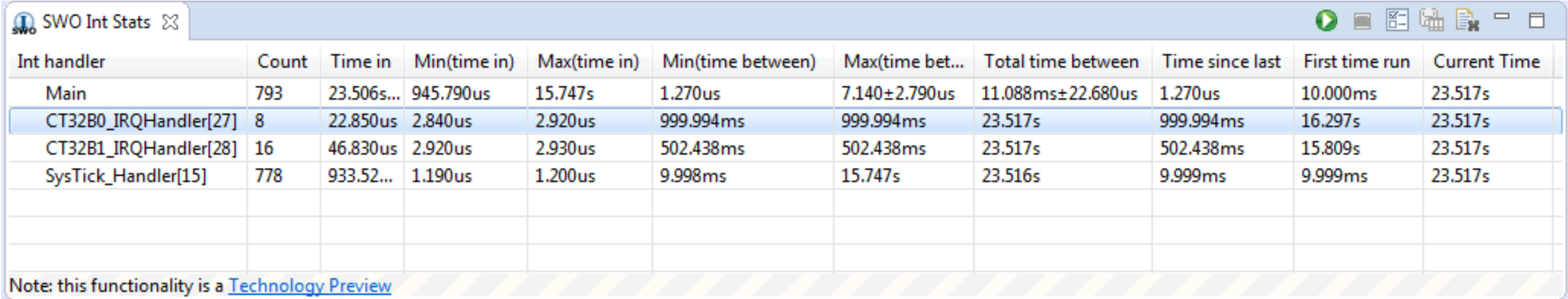
SWO Trace – Interrupt Views

Index	ID	Event	Handler	Time	Ticks
3098	35	EXIT	I2C1_IRQHand...	16.048s	3273746073
3097	35	ENTRY	I2C1_IRQHand...	16.048s	3273745862
3096	0	RETURN		16.048s±3.294us	3273740833
3095	-3	OVERFLOW	SWO Overflow	16.048s±3.294us	3273740833
3094	0	RETURN		16.048s±3.294us	3273740833
3093	15	EXIT	SysTick_Handler	16.048s	3273740833
3092	15	ENTRY	SysTick_Handler	16.048s	3273740817
3091	31	EXIT	TIMER3_IRQH...	16.048s	3273740810
3090	31	ENTRY	TIMER3_IRQH...	16.034s	3270940679



SWO Trace – Interrupt Views

- Interrupts Stats
 - Continuous count (and other stats) of all interrupts
- Benefits
 - Determine time spent in interrupt handlers
 - Optimization of interrupt handlers



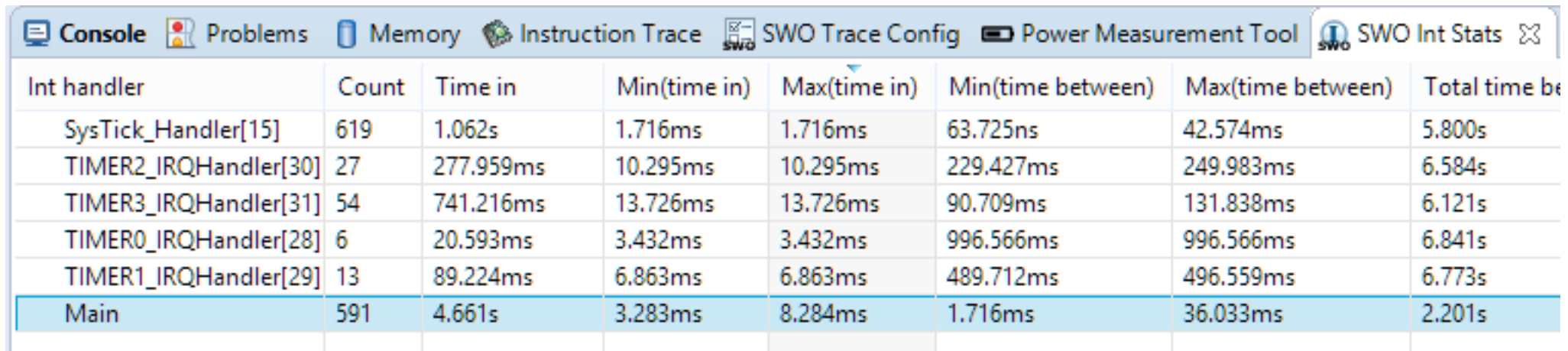
The screenshot shows a window titled "SWO Int Stats" with a table of interrupt statistics. The table has 11 columns: Int handler, Count, Time in, Min(time in), Max(time in), Min(time between), Max(time between), Total time between, Time since last, First time run, and Current Time. The data is as follows:

Int handler	Count	Time in	Min(time in)	Max(time in)	Min(time between)	Max(time between)	Total time between	Time since last	First time run	Current Time
Main	793	23.506s...	945.790us	15.747s	1.270us	7.140±2.790us	11.088ms±22.680us	1.270us	10.000ms	23.517s
CT32B0_IRQHandler[27]	8	22.850us	2.840us	2.920us	999.994ms	999.994ms	23.517s	999.994ms	16.297s	23.517s
CT32B1_IRQHandler[28]	16	46.830us	2.920us	2.930us	502.438ms	502.438ms	23.517s	502.438ms	15.809s	23.517s
SysTick_Handler[15]	778	933.52...	1.190us	1.200us	9.998ms	15.747s	23.516s	9.999ms	9.999ms	23.517s

Note: this functionality is a [Technology Preview](#)

SWO Interrupt Stats

- Capturing Interrupt Trace also populates an SWO Int Stats View
- **Switch to the SWO Int Stats View and compare the Stats with our measured values**
 - Note that for linear functions the Min (time in) and Max (time in) will be the same
 - this is not true for time spent in Main (which is the interrupted code)
 - Note the Min (time between) and Max (time between) is not guaranteed to be same
 - This relates to interrupt default priorities and pre-emption

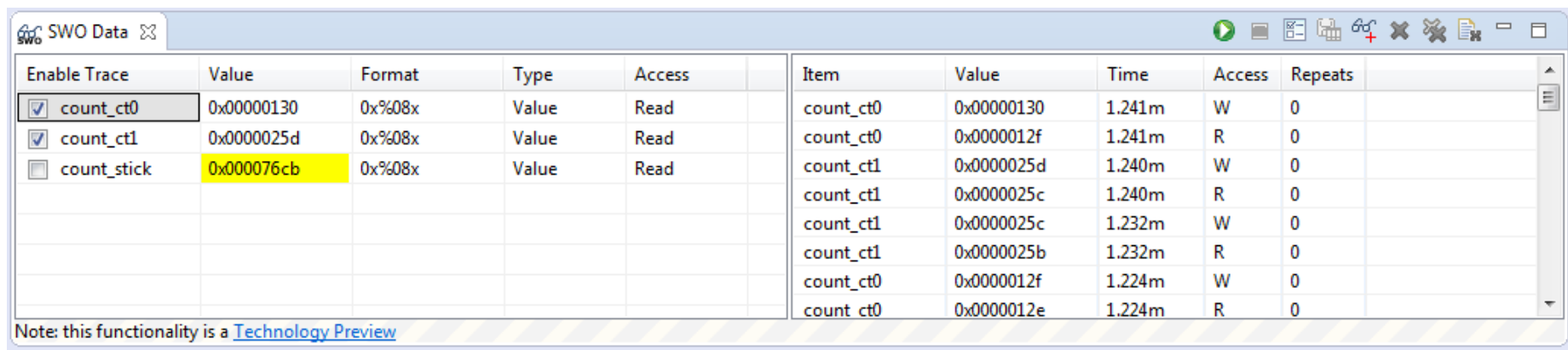


The screenshot shows the 'SWO Int Stats' view in a debugger. The table below represents the data shown in the screenshot.

Int handler	Count	Time in	Min(time in)	Max(time in)	Min(time between)	Max(time between)	Total time between
SysTick_Handler[15]	619	1.062s	1.716ms	1.716ms	63.725ns	42.574ms	5.800s
TIMER2_IRQHandler[30]	27	277.959ms	10.295ms	10.295ms	229.427ms	249.983ms	6.584s
TIMER3_IRQHandler[31]	54	741.216ms	13.726ms	13.726ms	90.709ms	131.838ms	6.121s
TIMER0_IRQHandler[28]	6	20.593ms	3.432ms	3.432ms	996.566ms	996.566ms	6.841s
TIMER1_IRQHandler[29]	13	89.224ms	6.863ms	6.863ms	489.712ms	496.559ms	6.773s
Main	591	4.661s	3.283ms	8.284ms	1.716ms	36.033ms	2.201s

SWO Trace – DataWatch

- Dynamic memory accesses
 - Read and write to target memory without stopping CPU
 - Non-intrusive
 - Reads done on periodic basis (by default)
 - Unlimited number of addresses
 - Allows modifications to parameters in real time
- Datawatch Trace
 - Capture all accesses to memory location, without stopping CPU
- Benefits
 - Monitor and analyse memory accesses
 - Identify ‘rogue’ memory accesses



The screenshot shows the 'SWO Data' window with two tables. The left table is for configuring DataWatch, and the right table shows the captured trace data.

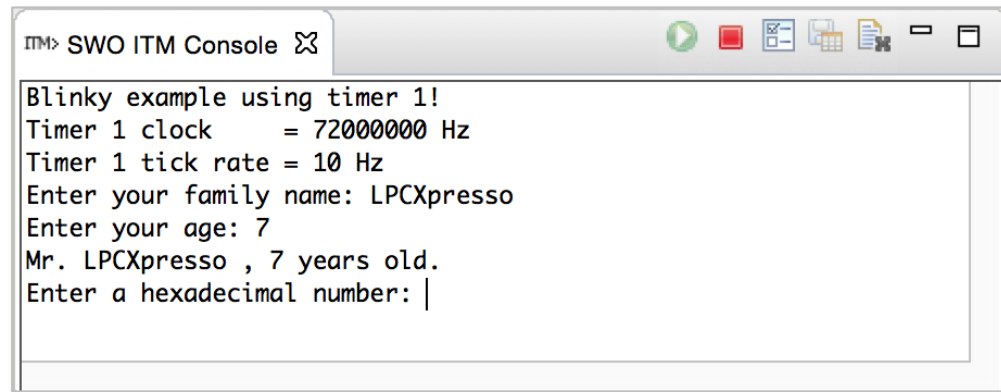
Enable Trace	Value	Format	Type	Access	
<input checked="" type="checkbox"/>	count_ct0	0x00000130	0x%08x	Value	Read
<input checked="" type="checkbox"/>	count_ct1	0x0000025d	0x%08x	Value	Read
<input type="checkbox"/>	count_stick	0x000076cb	0x%08x	Value	Read


Item	Value	Time	Access	Repeats
count_ct0	0x00000130	1.241m	W	0
count_ct0	0x0000012f	1.241m	R	0
count_ct1	0x0000025d	1.240m	W	0
count_ct1	0x0000025c	1.240m	R	0
count_ct1	0x0000025c	1.232m	W	0
count_ct1	0x0000025b	1.232m	R	0
count_ct0	0x0000012f	1.224m	W	0
count_ct0	0x0000012e	1.224m	R	0

Note: this functionality is a [Technology Preview](#)

SWO Trace – ITM Printf

- Instrumentation Trace Macrocell (ITM) block provides a mechanism for sending data from your target to the debugger via the SWO trace stream
- MCUXpresso IDE allows user to redirect printf/scanf data by reimplementing low level Redlib function `__sys_write` / `__sys_readc`
 - Newlib reimplementation also possible
- Unlike normal semihosting, this scheme is both low bandwidth and does not halt the MCU to transfer data



```
ITM> SWO ITM Console 
Blinky example using timer 1!
Timer 1 clock      = 72000000 Hz
Timer 1 tick rate = 10 Hz
Enter your family name: LPCXpresso
Enter your age: 7
Mr. LPCXpresso , 7 years old.
Enter a hexadecimal number: |
```