Freescale Semiconductor
Application Note

Document Number:AN4506
Rev. 0, 6/2012

Software Analysis in CodeWarrior for MCU

Basic Concepts

by: Razvan Ionescu
DevTech. Software Engineer
Bucharest
Romania

Contents

1 Introduction

Embedded engineers use features like debugger and compiler, and rarely use Software Analysis features like Trace, Profiler, or Code Coverage. However, those tools are less known but are as important and help a lot in a project. CodeWarrior for MCU provides an extensive set of Software Analysis (SA) tools, this application note explains and popularizes SA basic concepts. It will demonstrate how to enable Trace and Profile, how to collect trace, and how to read the results. ARM based processor from Freescale, Kinetis K70, and CodeWarrior for MCU v10.2 (b120126) are used to demonstrate SA capabilities.

2 Theory of Operation

There are some basic SA concepts that need to be explained before presenting main SA features in CodeWarrior.

2.1 Internal vs. external buffers

The buffer is the location where an executed trace is written by the hardware during the application. Most of the time, buffers are implemented using RAM memory. Buffers can be located on the analyzed device (internal) or on a trace collector device (external), which is usually used to perform the run-control function. CodeWarrior supports for Kinetis one internal buffer and two devices with external buffers (P&E TraceLink and Segger J-Trace).

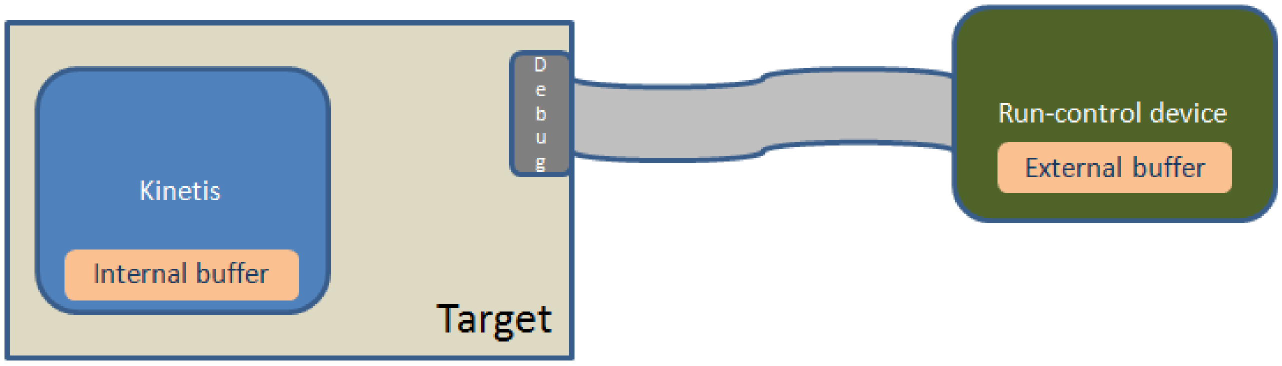


Figure 1. Internal and external buffers

For the Kinetis family, platform 2 and platform 3 type devices support internal buffers and complete external buffer implementation. The size of internal buffer is 2 kB. The internal buffer is called Embedded Trace Buffer (ETB). The size for external buffer is 128 MB (P&E TraceLink) or 8 MB (Segger J-Trace).

2.2 Hardware trace capabilities for Kinetis

The Kinetis family has support for program trace (Embedded Trace Macrocell (ETM) and instrumentation trace (Instrument Trace Macrocell (ITM) for platform 2 and 3 devices. Platform 0 and 1 devices has support only for ITM trace.

The program trace (ETM) provides information about executed code in a compressed format that allows, after an offline post-processing, to fully reconstruct the application execution. It is possible to use methods to limit the amount of program trace generated by the core.

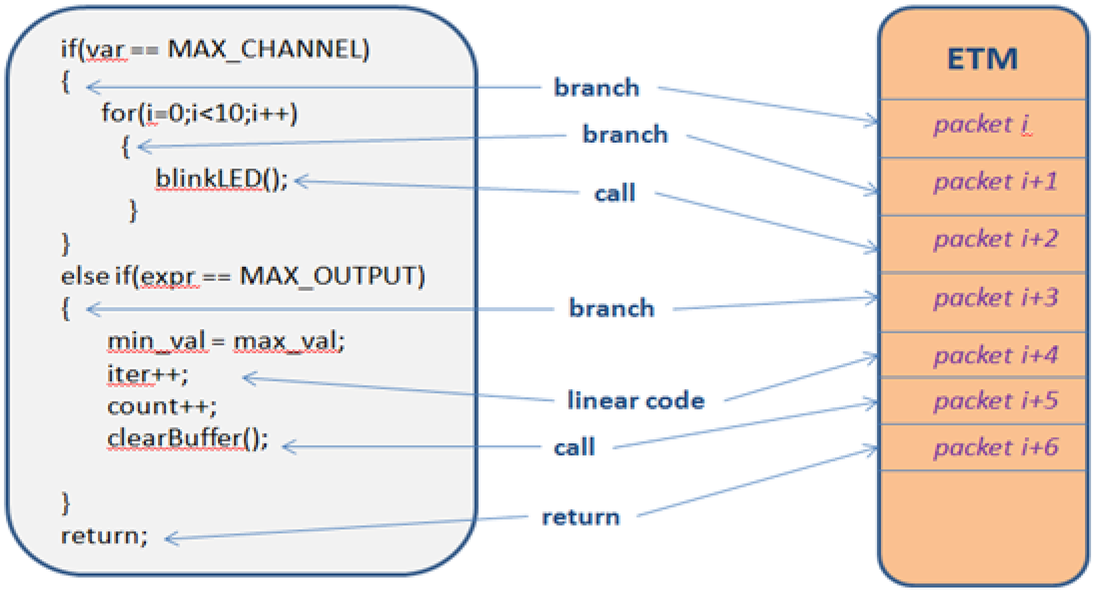

```
if(var == MAX_CHANNEL)
{
    for(i=0;i<10;i++)
     {
        blinkLED();
     }
}
else if(expr == MAX_OUTPUT)
{
    min_val = max_val;
    iter++;
    count++;
    clearBuffer();

}
return;
```

Figure 2. Program trace (ETM)

Instrumentation trace (ITM) streams information from two sources: application code and Data Watchpoint and Trace(DWT) unit. A developer can insert into application code (to instrument) that writes to certain registers with custom data. That data is inserted into the ITM trace in raw format. From the DWT, some kind of data trace is generated. Performance counters values are also sampled into the ITM trace.

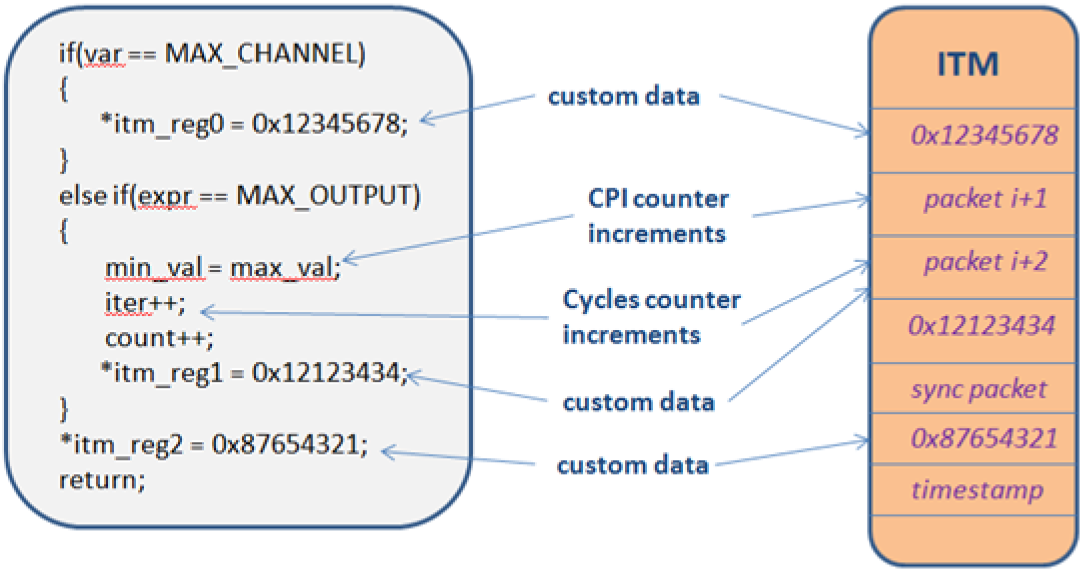


Figure 3. Instrumentation trace (ITM)

The Trace Port Interface Unit (TPIU) and Embedded Trace Buffer (ETB) are formatters, and package the trace. Trace funnel has the role of mixing the trace streams from the ETM and ITM in one stream.

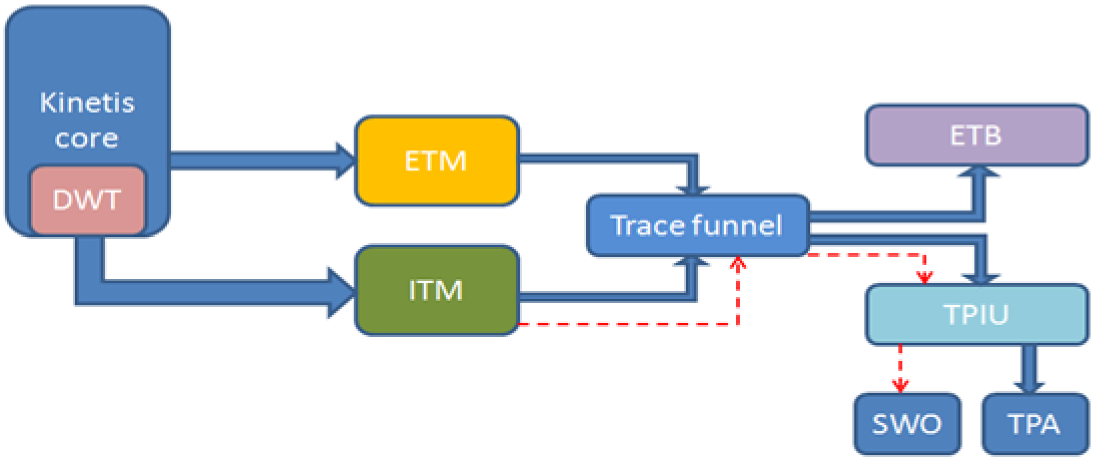


Figure 4. Trace formatters—ETB and TPIU

NOTE

Kinetis does not allow to send trace simultaneously to the ETB and TPIU. Only one path can be used at once.

2.3 Timestamps

There are 2 timestamp counters for Kinetis: a global 48-bits counter and a local 21-bits counter. The global counter can be used to put timetamps on events in ETM and ITM trace streams. By doing this, you have a synchronization of events from the 2 streams, that is because the clock is the same. The local timestamp counter can be used only by ITM events.

Also note that the global timestamp counter does not stop while you are in debug mode, so the time spent in debug mode is also recorded (for example, breakpoints, and prints). The local timestamp counter is frozen during debug mode, therefore it is counting only the time spent in execution mode.

2.4 Trace collection modes

There are three types of collecting trace modes:

- **Overwrite**

  In this mode, when trace events fill the trace buffer, the first events (oldest) are overwritten my the last events (more recent). At the end of the collection, the trace buffer will have the last n events, where n is trace buffer capacity (calculated in events). Think of this as a circular buffer.

  This mode is useful for when the user investigates an extreme condition (for example, a crash or an exception), because in trace the last event is found that generated that condition.

  A special attribute of this mode is that its the least intrusive—basically trace IP monitors core/platform execution, and record events, but do not interfere with execution.

  This mode can be used with ETB (internal trace buffer) or with an external buffer (P&E TraceLink or Segger J-Trace).

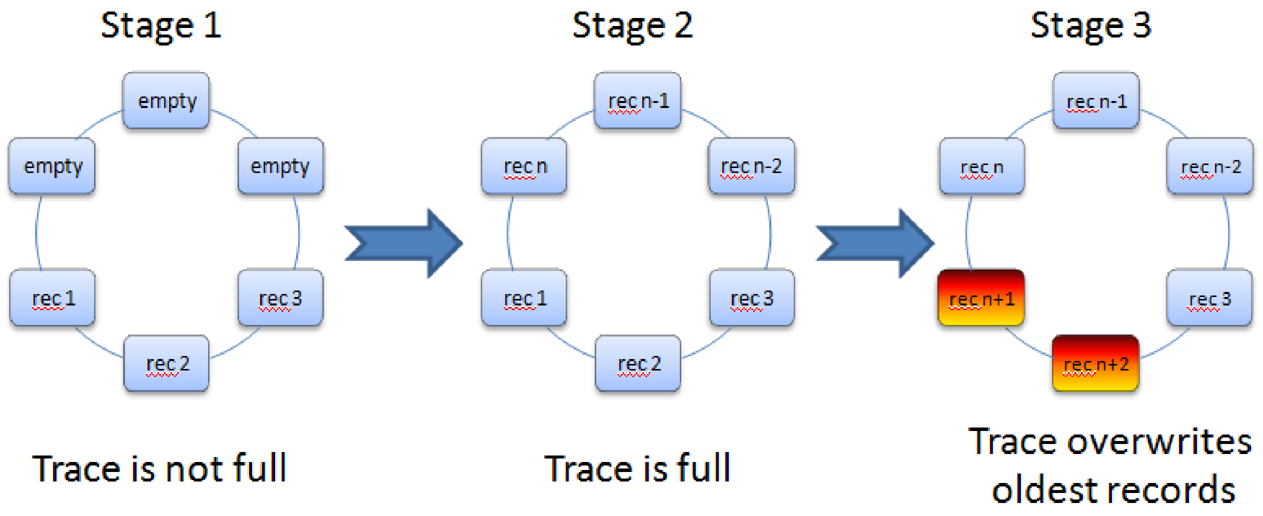


Figure 5. Overwrite trace

- **Continuous**

  In this mode when the trace buffer is full the core stops execution and enters in debug mode. CodeWarrior detects debug status of the target and collect trace and then resumes execution. This mode assures that all execution is traced and kept and nothing is lost. However, it is intrusive and may break the real-time nature of an application.

  This mode is useful when a Profiler is needed (to gather enough data to make the Profiler useful). Also it is useful in case of Code Coverage analysis, where real-time is not a strong requirement.

  This mode can be used only with the internal trace buffer (ETB).

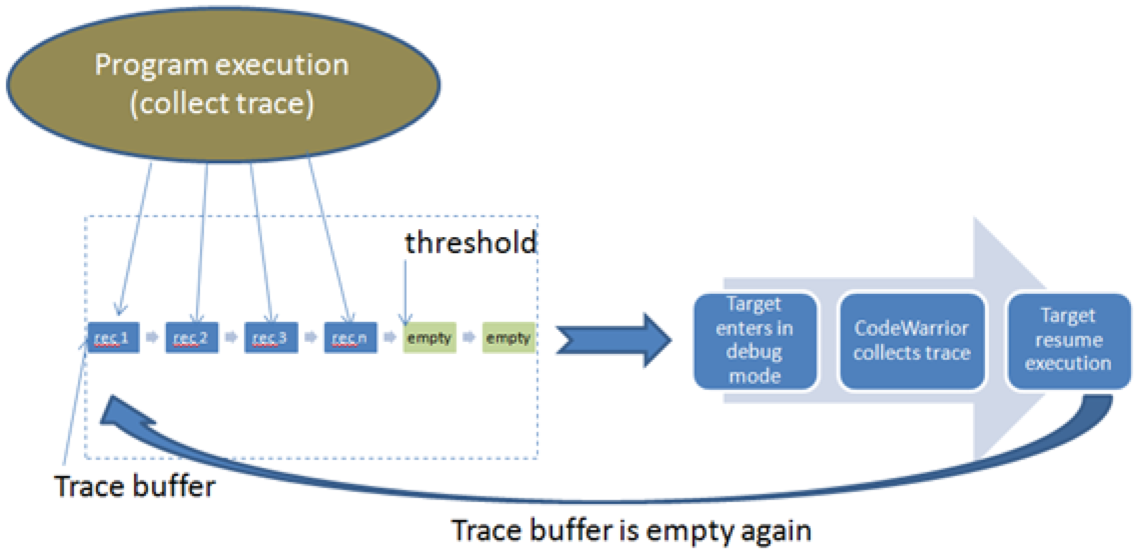

Program execution (collect trace)
threshold
rec 1
rec 2
rec 3
rec n
empty
empty
Trace buffer
Target enters in debug mode
CodeWarrior collects trace
Target resume execution
Trace buffer is empty again

Figure 6. Continuous trace

- **Pseudo-continuous**
  This mode is a combiation of the first two. It consists of collecting trace with an external device (P&E TraceLink or Segger J-Trace) and keeps all buffers. However, if an external buffer (which runs by design in overwrite mode) overflows, then a gap appears in trace.

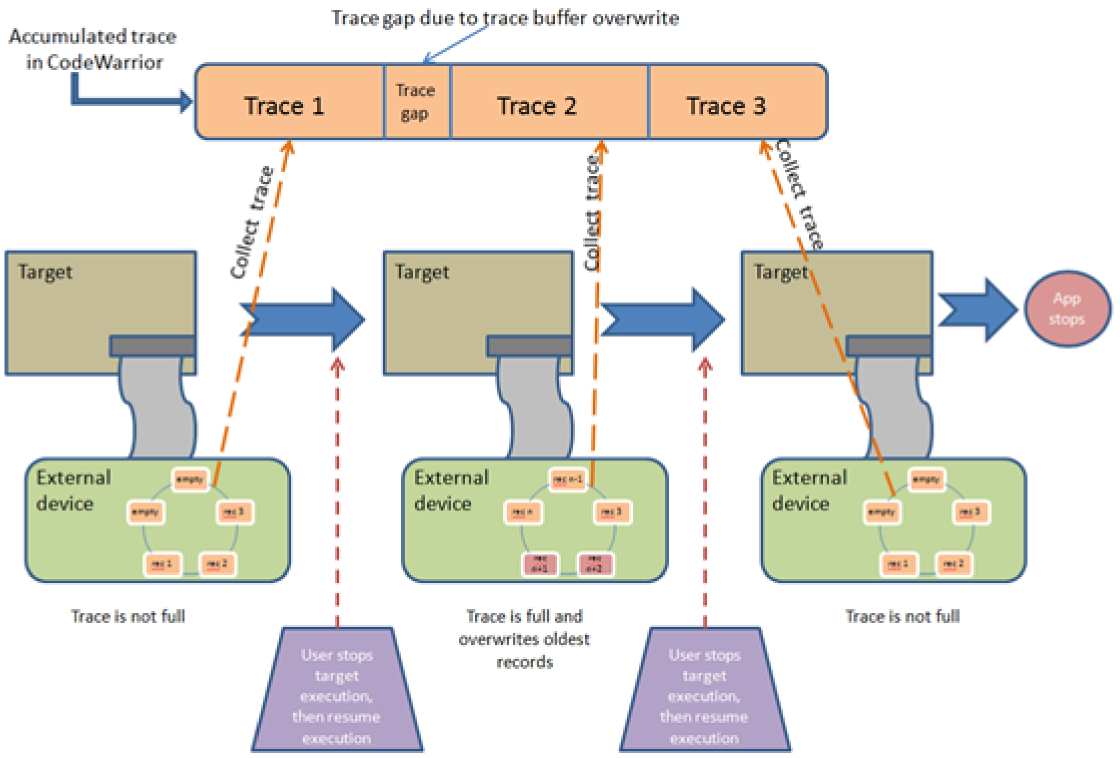

Accumulated trace in CodeWarrior
Trace gap due to trace buffer overwrite
Trace 1
Trace gap
Trace 2
Trace 3
Collect trace
Collect trace
Collect trace
Target
Target
Target
App stops
External device
External device
External device
Trace is not full
Trace is full and overwrites oldest records
Trace is not full
User stops target execution, then resume execution
User stops target execution, then resume execution

Figure 7. Pseudo-continuous trace

2.5 Profiling counters

Kinetis supports five 8-bits profiling counters and one 32-bits cycles counter:

- CPI counter (8-bit):

  This is the general counter for instruction cycle count estimation. It increments on any additional cycles (the first cycle is not counted) required to execute multi-cycle instructions except those recorded by LSU counter. It also increments on any instruction fetch stalls.

- LSU counter (8-bit):

  It is the load-store counter. The counter increments on any additional cycles (the first cycle is not counted) required to execute multi-cycle load-store instructions.

- FOLD counter (8-bit):

  It is the folded instruction counter. The counter increments on any instruction that executes in zero cycles.

- Exception counter (8-bit):

  It is the exception overhead counter. The counter increments on any cycles associated with exception entry and return.

- SLEEP counter (8-bit):

  It is the sleep overhead counter. The counter increments on any cycles associated with power saving, initiated by the WFI or WFE instructions.

- CYCLES counter (32-bits):

  It is the cycles counter.

2.6 Single Wire Output (SWO) support

Single Wire Output (SWO) is a protocol that outputs the ITM trace through a separate path. This is basically a low-cost debug and trace method. The bandwidth is limited, but enough information can be sent to create a big picture of application execution. This type of trace collection does not need of a trace buffer (internal or external) and uses a streaming approach. All data is sent directly to the host machine.

This method of tracing is not intrusive.

NOTE

SWO can be used only over SWD protocol (not over JTAG), therefore make sure debugger protocol is properly set.

2.7 How to instrument an application

Kinetis provides 32 stimulus registers. When data is written to one of these registers, it is automatically inserted in trace. A common code that maps a register and writes data is:

```
volatile unsigned long *itm_stim_0 = (unsigned long *) 0xE0000000;
…
*itm_stim_0 = 0x11223344;
…
*itm_stim_0 = 0x55667788;
```

2.8 Trace protocol synchronization packets

Both ETM and ITM trace streams implement a set of synchronization packets that allow to recover after a special condition (for example, trace lost, and trace buffer overflows).

Synchronization packets can be:

- Periodic (ETM):

  After each 1024 bytes entered in ETM trace, a sync packet is sent. This is useful, for example, when you have trace collected using a circular buffer, ETB or external device, in overwrite mode. After the buffer is full, the oldest records are overwritten with the most recent ones. In this case, it will overwrite sync packets (sent at the start of trace) and the stream will be un-decodable. After 1024 bytes, a periodic ETM sync packet is sent, the stream will then be correctly decoded from that point.

- Periodic (ITM):

  The default value for the ITM sync packet is 1024 bytes. This can be adjusted. Please see the above description to see how the sync packet helps recreate the execution flow.

- Address sync (ETM):

  Even if a periodic sync is emitted by the ETM from 1024 to 1024 bytes, this is not enough to recover from a trace lost or from a trace buffer overwrite. The stream is decoded, but ETM compression mechanism for code addresses is made with relative offset. In other words, after the ETM places a packet in address 0x100, if the next executed address is 0x104, the ETM codes the address as 0x4. This allows to save bandwidth, but generates some wrong addresses to be decoded after a special condition event (for example trace lost, and trace buffer overwrite). But an Address-sync packet will sample a full address value into the trace. From that point, both decoding and trace recreation is correctly executed.

- Timestamp sync (ETM):

  As for instruction addresses (see above), the timestamp value is also coded using a relative offset. So just after a timestamp sync, the time reported in trace is accurate after a special condition event (for example, trace lost, and trace buffer overwrite).

- General trace sync (ETM, ITM):

  There is a special packet that differentiates the start of the trace from noise. This is put in trace on events like: start trace, exit from debug, or at other events that may trigger a trace start.

- Trigger packets (ETM, ETB):

  Are not sync packets from a protocol perspective, but can be viewed as logical sync points. It marks the tracing triggering point.

2.9 Extra trace features in Kinetis

There are some other trace, or trace related capabilities in Kinetis, but they are not the object of this application note.

3 Practical Content

This section indicates how to import a project into CodeWarrior, enable Trace & Profile, instrument the code, run application on K70 target, collect trace and see the results. All trace collection modes are shown (overwrite, continuous, and pseudo-continuous). The SWO collection is also shown.

1. Choose trace collection mode: overwrite, continuous, pseudo-continuous, and SWO
2. Run application (collect trace progress monitor)
3. SA view
4. Trace
5. Critical code and coverage
6. Performance and call tree
7. Log
8. Repeat c-i for all trace collection modes

3.1 Import the demo project

1. Open CodeWarrior and right-click in the projects area.

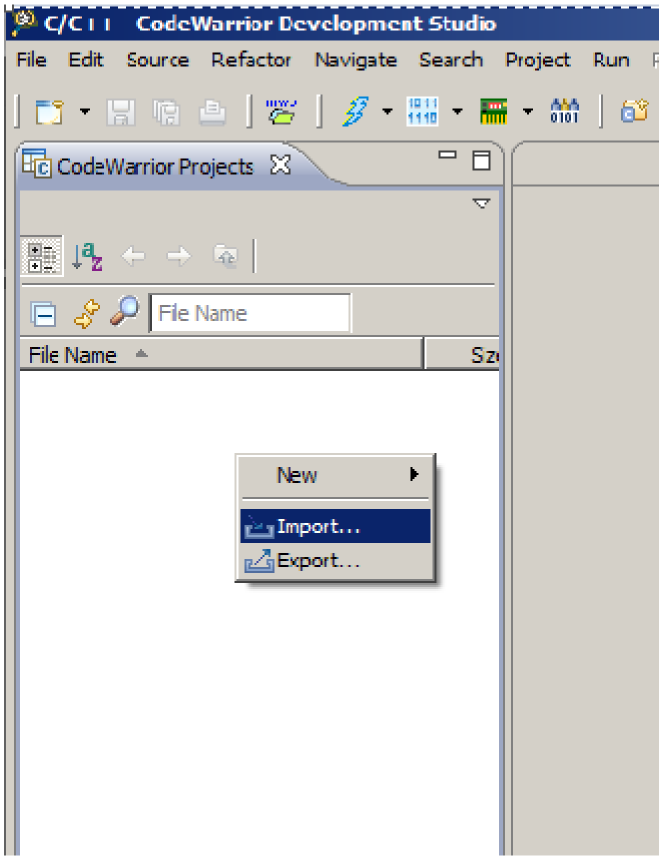

C/C++ - CodeWarrior Development Studio
File Edit Source Refactor Navigate Search Project Run
CodeWarrior Projects
File Name
File Name
Siz
New
Import...
Export...

Figure 8. Import menu

2. Click on Import, then choose to open an existing CodeWarrior project:

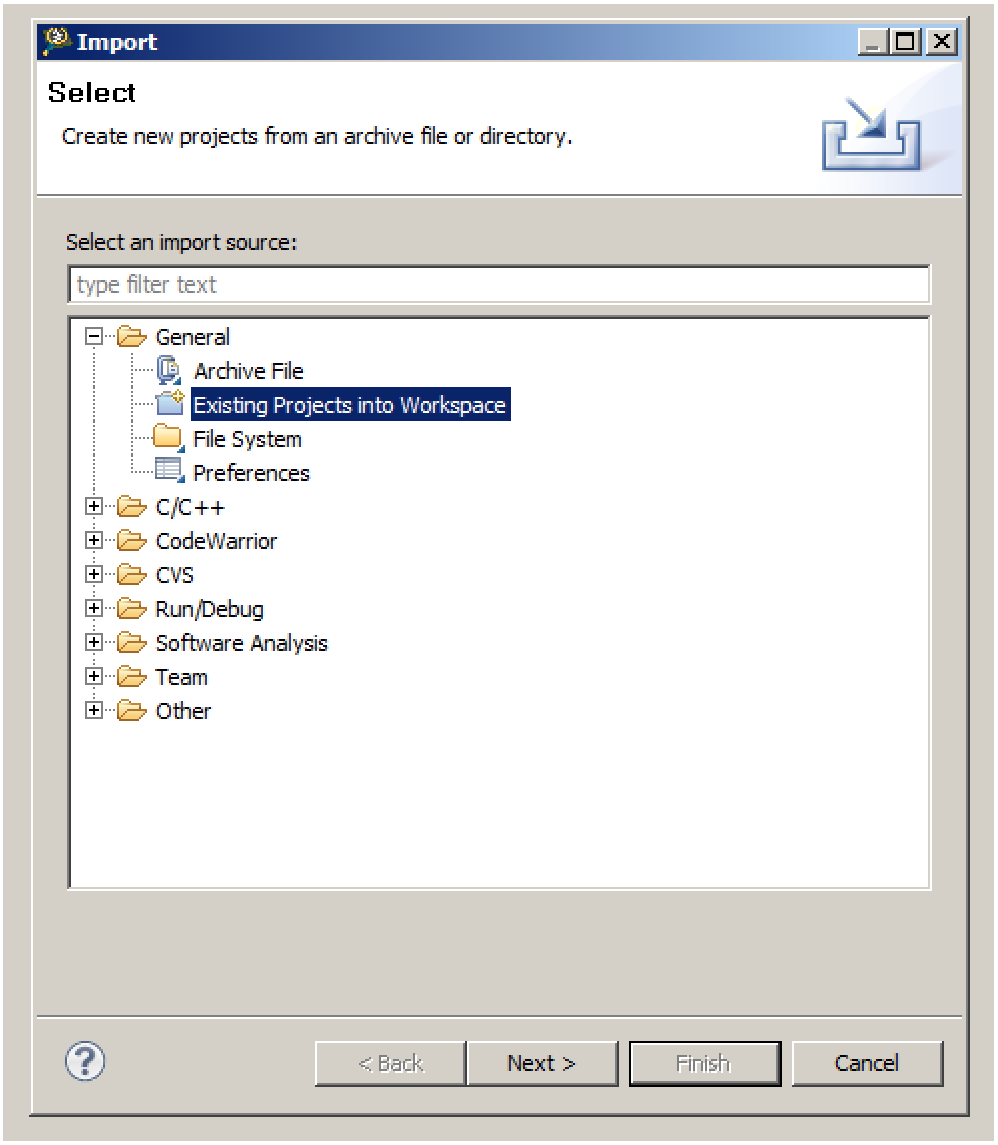


Figure 9. Opening an existing project

3. Browse to the project location:

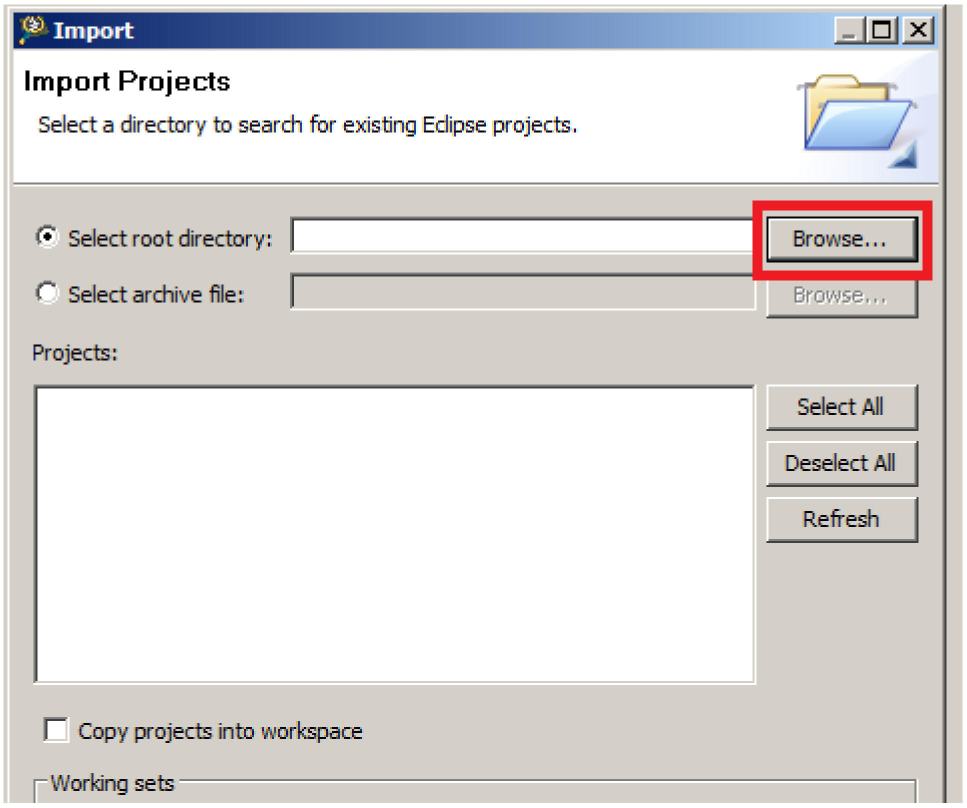
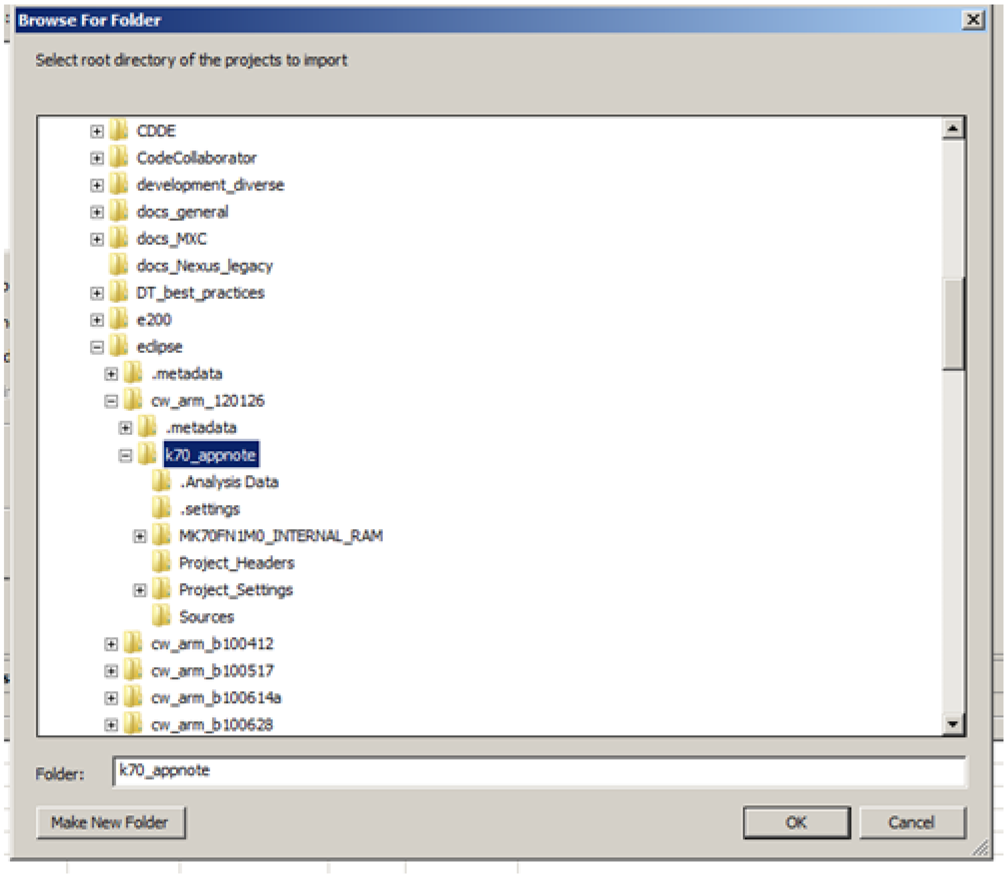


Figure 10. Browse to project location

4. Select project's directory:


Browse For Folder

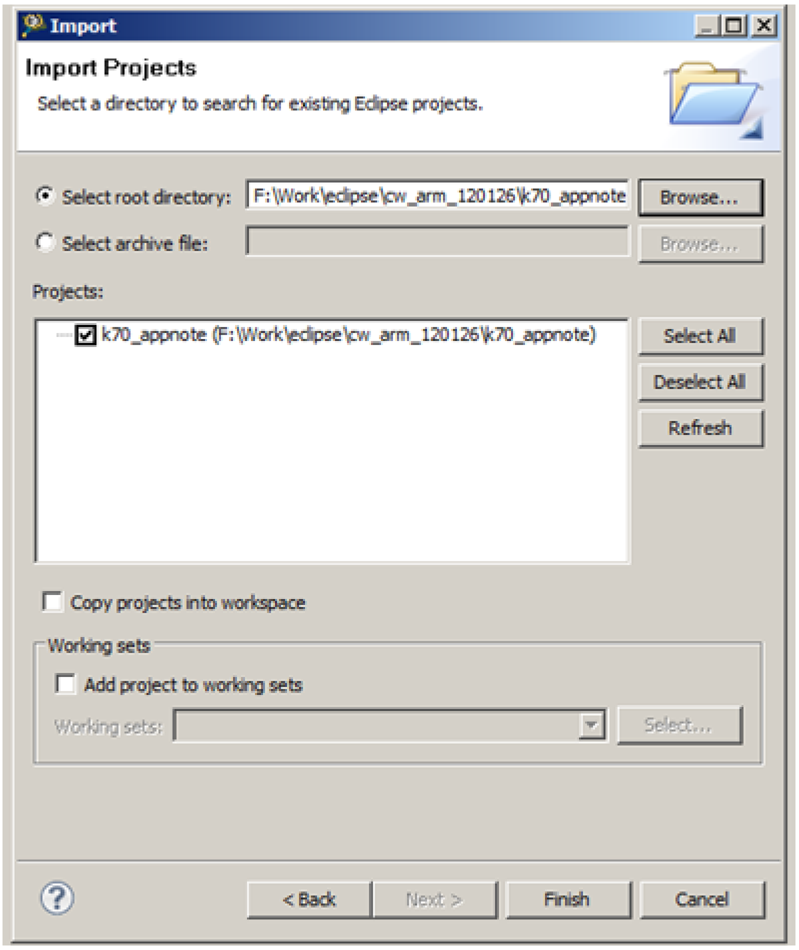
Select root directory of the projects to import

- CDDE
- CodeCollaborator
- development_diverse
- docs_general
- docs_MXC
- docs_Nexus_legacy
- DT_best_practices
- e200
- eclipse
  - .metadata
  - cw_arm_120126
    - .metadata
    - k70_appnote
      - .Analysis Data
      - .settings
      - MK70FN1M0_INTERNAL_RAM
      - Project_Headers
      - Project_Settings
      - Sources
  - cw_arm_b100412
  - cw_arm_b100517
  - cw_arm_b100614a
  - cw_arm_b100628

Folder: k70_appnote

Figure 11. Select project’s directory

5. Notice the project is recognized by CodeWarrior; click Finish:



Select root directory: F:\Work\eclipse\cw_arm_120126\k70_appnote
Select archive file:

Projects:

[✓] k70_appnote (F:\Work\eclipse\cw_arm_120126\k70_appnote)

[ ] Copy projects into workspace

Working sets

[ ] Add project to working sets

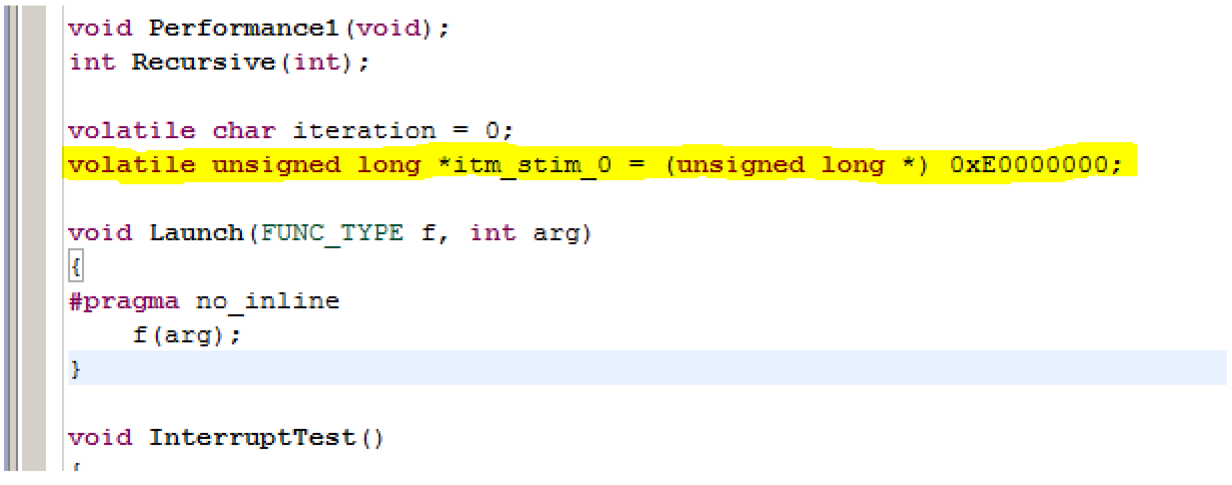
Working sets:

Figure 12. Finish the import

6. Build the project.

3.2 Instrument the code

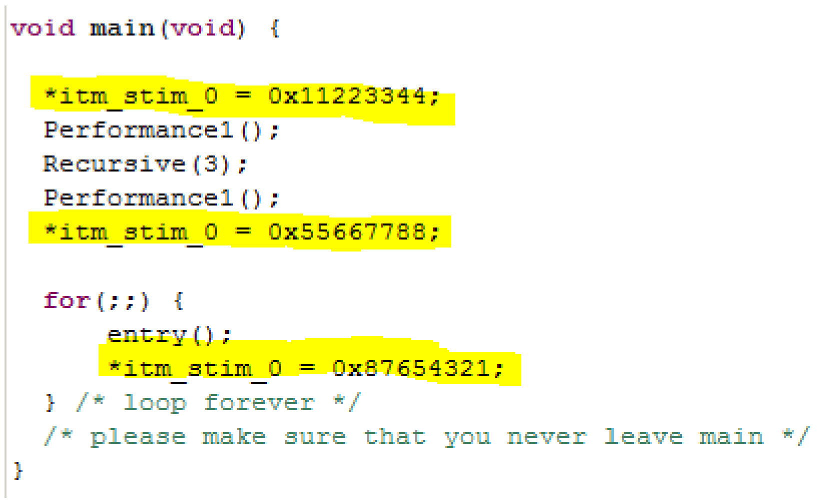
Open main.c file and edit as below:



```
void Performance1(void);
int Recursive(int);

volatile char iteration = 0;
volatile unsigned long *itm_stim_0 = (unsigned long *) 0xE0000000;

void Launch(FUNC_TYPE f, int arg)
{
#pragma no_inline
    f(arg);
}

void InterruptTest()
```

Figure 13. Edit main.c file – define stimulus port

Define the stimulus port to write the custom data. Then write some custom data from code:

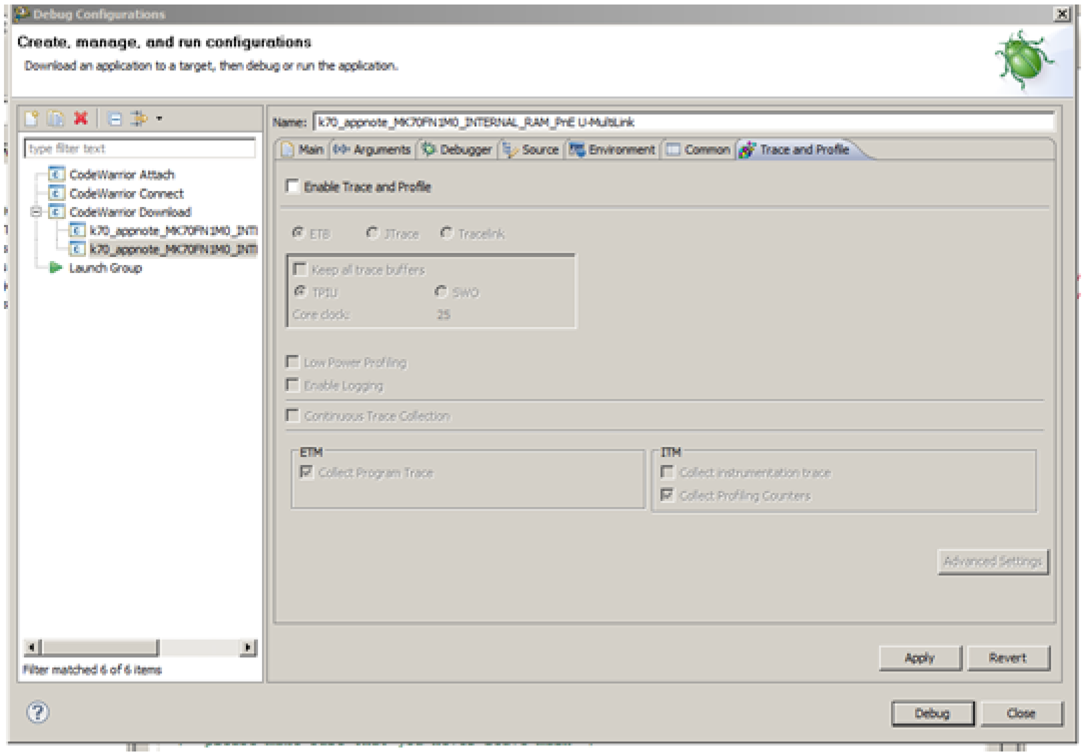


```
void main(void) {

  *itm_stim_0 = 0x11223344;
  Performance1();
  Recursive(3);
  Performance1();
  *itm_stim_0 = 0x55667788;

  for(;;) {
      entry();
      *itm_stim_0 = 0x87654321;
  } /* loop forever */
  /* please make sure that you never leave main */
}
```

Figure 14. Edit main.c file – write to stimulus port

3.3 Enable Trace and Profile

Open Debug configurations for imported project and select tab *Trace and Profile* for k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink configuration.



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

Tabs: Main | Arguments | Debugger | Source | Environment | Common | Trace and Profile

- type filter text
- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
  - k70_appnote_MK70FN1M0_INTI
  - k70_appnote_MK70FN1M0_INTI
- Launch Group

Filter matched 6 of 6 items

☐ Enable Trace and Profile

(●) ETB ( ) JTrace ( ) Tracelink

☐ Keep all trace buffers

(●) TPIU ( ) SWO

Core clock: 25

☐ Low Power Profiling

☐ Enable Logging

☐ Continuous Trace Collection

ETM

☑ Collect Program Trace

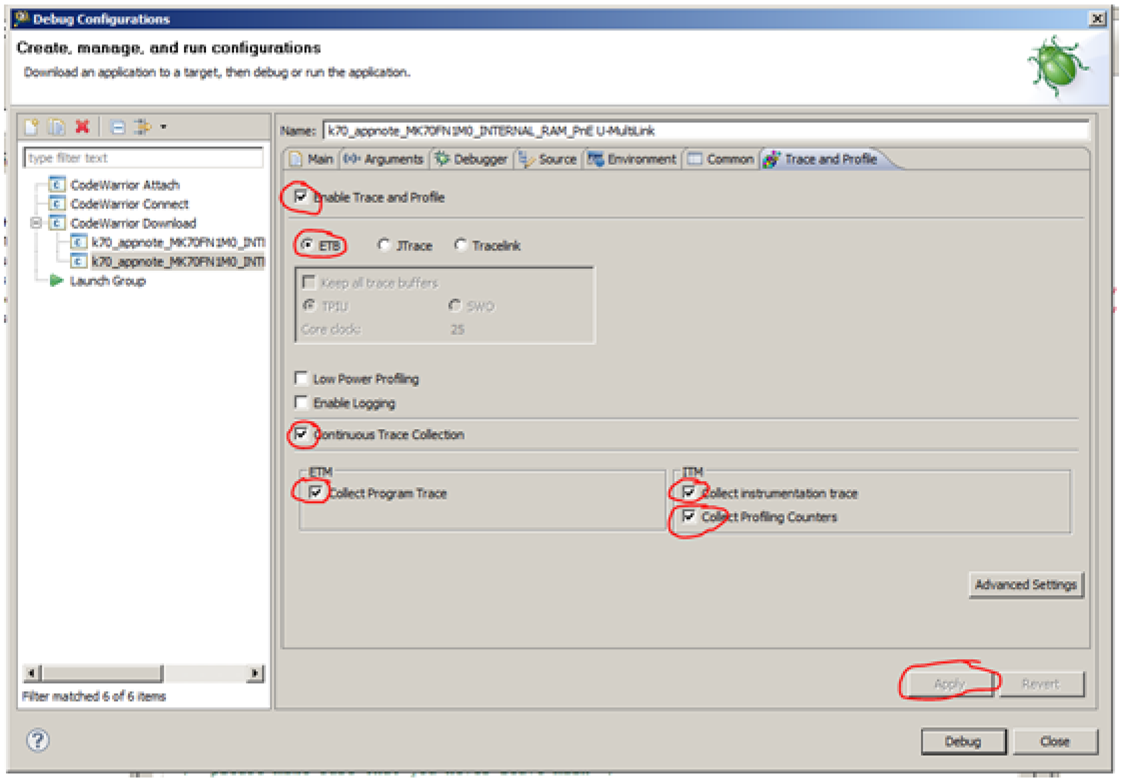
ITM

☐ Collect instrumentation trace

☑ Collect Profiling Counters

Figure 15. Select Trace and Profile tab

Check Enable Trace and Profile and make sure ETB, Continuous Trace Collection, Collect Program Trace, Collect instrumentation trace, and Collect Profiling Counters option is checked.



Create, manage, and run configurations

Download an application to a target, then debug or run the application.

type filter text

- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
  - k70_appnote_MK70FN1M0_INTI
  - k70_appnote_MK70FN1M0_INTI
- Launch Group

Filter matched 6 of 6 items

Name: k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

[✓] Enable Trace and Profile

(●) ETB ( ) JTrace ( ) Tracelink

[ ] Keep all trace buffers

(●) TPIU ( ) SWO

Core clock: 25

[ ] Low Power Profiling

[ ] Enable Logging

[✓] Continuous Trace Collection

ETM

[✓] Collect Program Trace

ITM

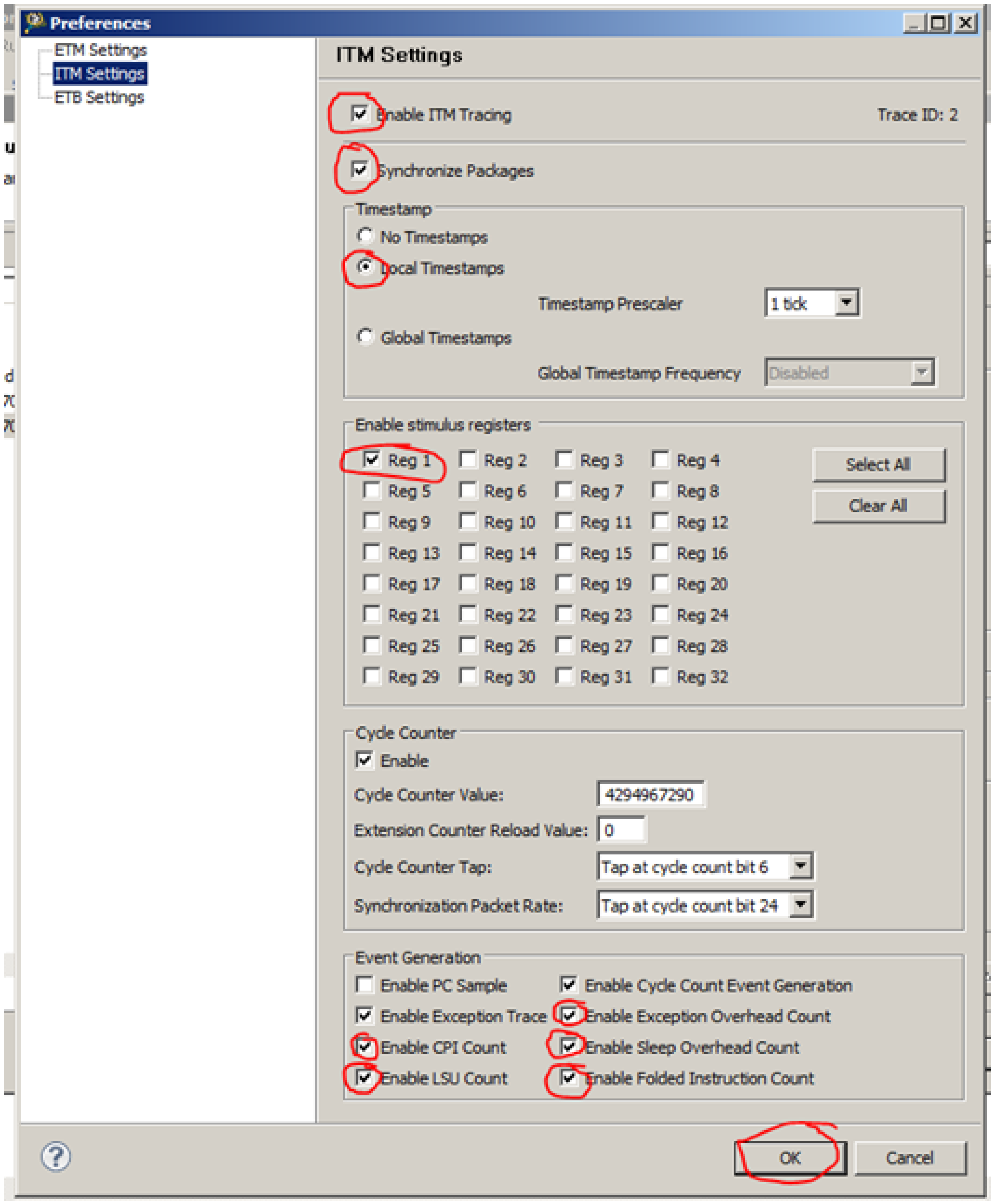
[✓] Collect instrumentation trace

[✓] Collect Profiling Counters

Figure 16. Enable Trace

Click *Apply* to confirm settings.

Click on *Advanced Settings* to see some more options:



Preferences

ETM Settings
ITM Settings
ETB Settings

ITM Settings

[✓] Enable ITM Tracing   Trace ID: 2

[✓] Synchronize Packages

Timestamp
- ( ) No Timestamps
- (●) Local Timestamps
  - Timestamp Prescaler: 1 tick
- ( ) Global Timestamps
  - Global Timestamp Frequency: Disabled

Enable stimulus registers

[✓] Reg 1	[ ] Reg 2	[ ] Reg 3	[ ] Reg 4
[ ] Reg 5	[ ] Reg 6	[ ] Reg 7	[ ] Reg 8
[ ] Reg 9	[ ] Reg 10	[ ] Reg 11	[ ] Reg 12
[ ] Reg 13	[ ] Reg 14	[ ] Reg 15	[ ] Reg 16
[ ] Reg 17	[ ] Reg 18	[ ] Reg 19	[ ] Reg 20
[ ] Reg 21	[ ] Reg 22	[ ] Reg 23	[ ] Reg 24
[ ] Reg 25	[ ] Reg 26	[ ] Reg 27	[ ] Reg 28
[ ] Reg 29	[ ] Reg 30	[ ] Reg 31	[ ] Reg 32

Select All   Clear All

Cycle Counter
[✓] Enable
Cycle Counter Value: 4294967290
Extension Counter Reload Value: 0
Cycle Counter Tap: Tap at cycle count bit 6
Synchronization Packet Rate: Tap at cycle count bit 24

Event Generation

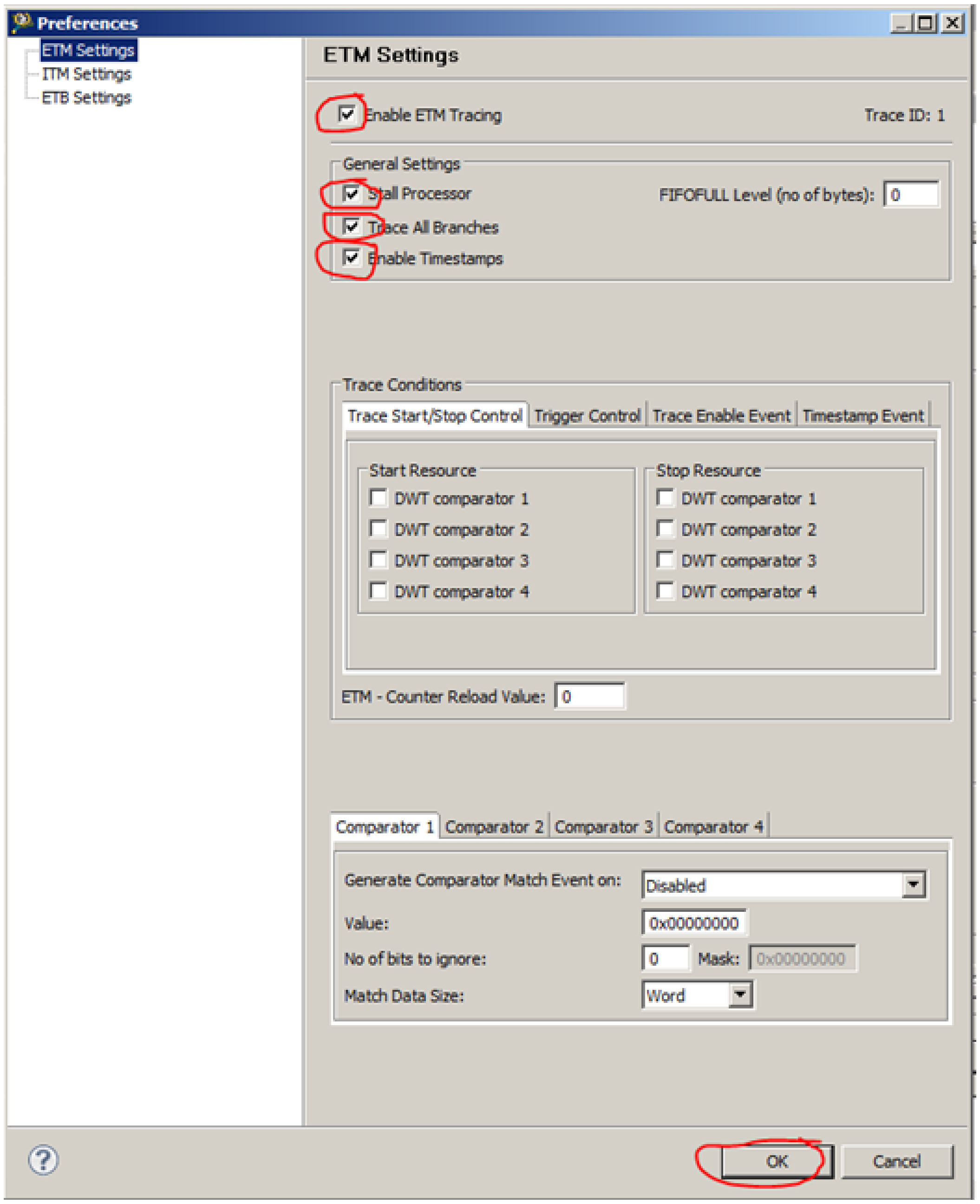
[ ] Enable PC Sample	[✓] Enable Cycle Count Event Generation
[✓] Enable Exception Trace	[✓] Enable Exception Overhead Count
[✓] Enable CPI Count	[✓] Enable Sleep Overhead Count
[✓] Enable LSU Count	[✓] Enable Folded Instruction Count

OK   Cancel

Figure 17. Enable and configure ITM tracing

Notice that ITM tracing is enabled. Synchronized packages are default on, because the bandwidth limitation is negligible to adding them and the benefit is huge. This helps synchronize the trace in case of a special condition event (for example, trace lost, and trace buffer overwrite). Notice that Local timestamp is used for this example.

Here onlt 1 stimulus register (from the application, write only at this register) is enabled. Also note the Performance counters enabled (section Event Generation). This triggers counter values to be sampled in trace.


Preferences

ETM Settings
ITM Settings
ETB Settings

ETM Settings

☑ Enable ETM Tracing   Trace ID: 1

General Settings
☑ Stall Processor   FIFOFULL Level (no of bytes): 0
☑ Trace All Branches
☑ Enable Timestamps

Trace Conditions
Trace Start/Stop Control | Trigger Control | Trace Enable Event | Timestamp Event

Start Resource	Stop Resource
☐ DWT comparator 1	☐ DWT comparator 1
☐ DWT comparator 2	☐ DWT comparator 2
☐ DWT comparator 3	☐ DWT comparator 3
☐ DWT comparator 4	☐ DWT comparator 4

ETM - Counter Reload Value: 0

Comparator 1 | Comparator 2 | Comparator 3 | Comparator 4

Generate Comparator Match Event on: Disabled
Value: 0x00000000
No of bits to ignore: 0   Mask: 0x00000000
Match Data Size: Word

OK   Cancel

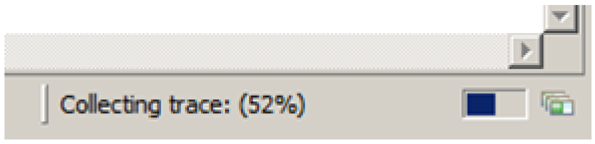
Figure 18. Enable and configure ETM tracing

For ETM advanced settings, notice the Stall processor option. This freezes the core activity when the internal ETM buffer is full to not lose trace. However, this function is limited because of many reasons, however it is better to have this option on by default. The trace all branches option configures the ETM trace to record all branches or just indirect branches. Theoretically, it is enough to sample only indirect branches, because other branches can be determined offline, just disassembling the binary executable. It is a method to save bandwidth. However, if you don't have overflows in trace due to bandwidth capacity, it is advisable to check this option for better decoding results.

Enable timestamps option configures trace to sample or not the timestamp. It is another method for saving bandwidth, however it means that trace will not have timestamps, so some analysis, like performance or critical code cannot be performed. If you want to put more code execution information in trace and you are not interesting in the timestamp, this option saves bandwidth.

3.4 Collect trace

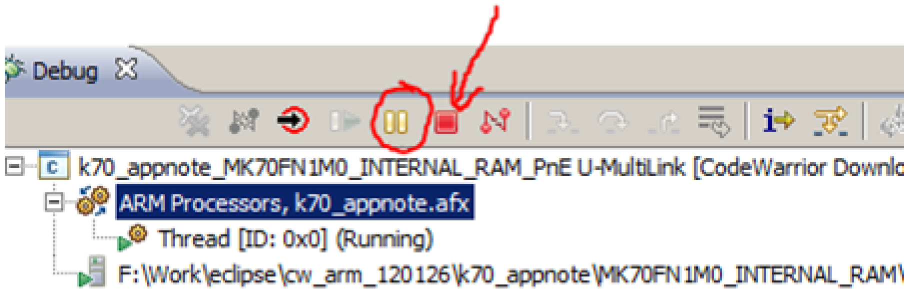
Enter debug mode and resume application. Notice that the current settings (continuous mode), the trace module stops the target execution when the internal buffer hits threshold, collect trace, and then resumes execution. In the bottom-right corner of CodeWarrior, look at the progress monitor for collecting trace operation.



Collecting trace: (52%)

Figure 19. Observe collection of trace progress monitor

The debugger has two buttons that control execution stop: Suspend (circled in picture below) and Terminate (indicated by an arrow in picture below).



```
k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink [CodeWarrior Downlo
  ARM Processors, k70_appnote.afx
    Thread [ID: 0x0] (Running)
  F:\Work\eclipse\cw_arm_120126\k70_appnote\MK70FN1M0_INTERNAL_RAM\
```

Figure 20. Debugger controls

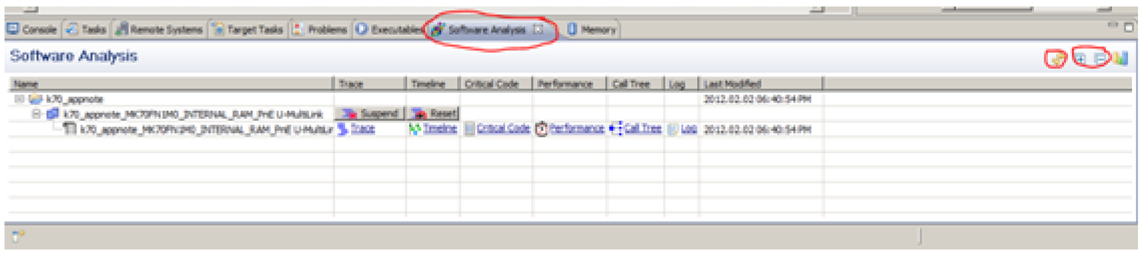
When Terminating an application, means that you want to exit immediately, and trace will not be collected then. However, for continuous mode, you will have trace collected up to the last collection moment. For overwrite mode (where trace is collected only when the application is suspended) you will have no trace.

In this case, Suspend and then Terminate application.

The Software Analysis view for results opens automatically.

3.5 Software analysis view

Software Analysis view opens automatically after trace collection. However, if you want to open this view on demand, go to the CodeWarrior menu: Window->Show view-> Software Analysis.



Name	Trace	Timeline	Critical Code	Performance	Call Tree	Log	Last Modified
k70_appnote							2012.02.02 06:40:54 PM
k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink	Suspend	Reset					
k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLir	Trace	Timeline	Critical Code	Performance	Call Tree	Log	2012.02.02 06:40:54 PM

Figure 21. Software Analysis view

Notice buttons on the upper-right side of the view. You can refresh the view, expand or collapse results.

Right-clicking on a result set, you can delete or save it.

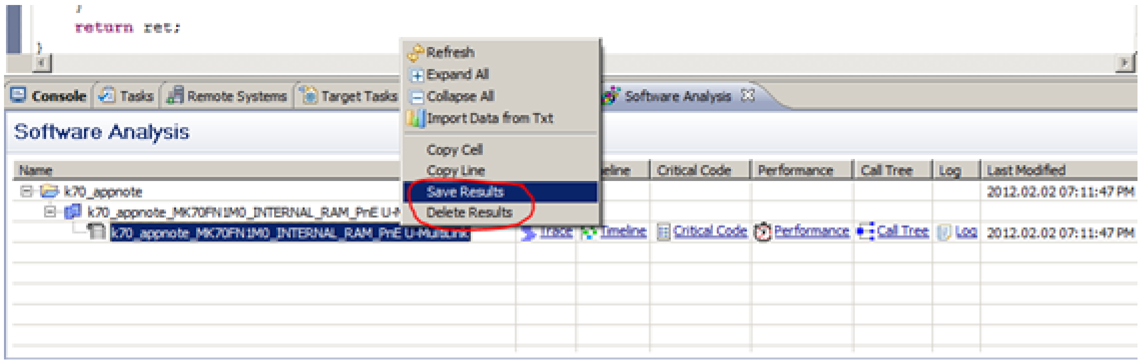
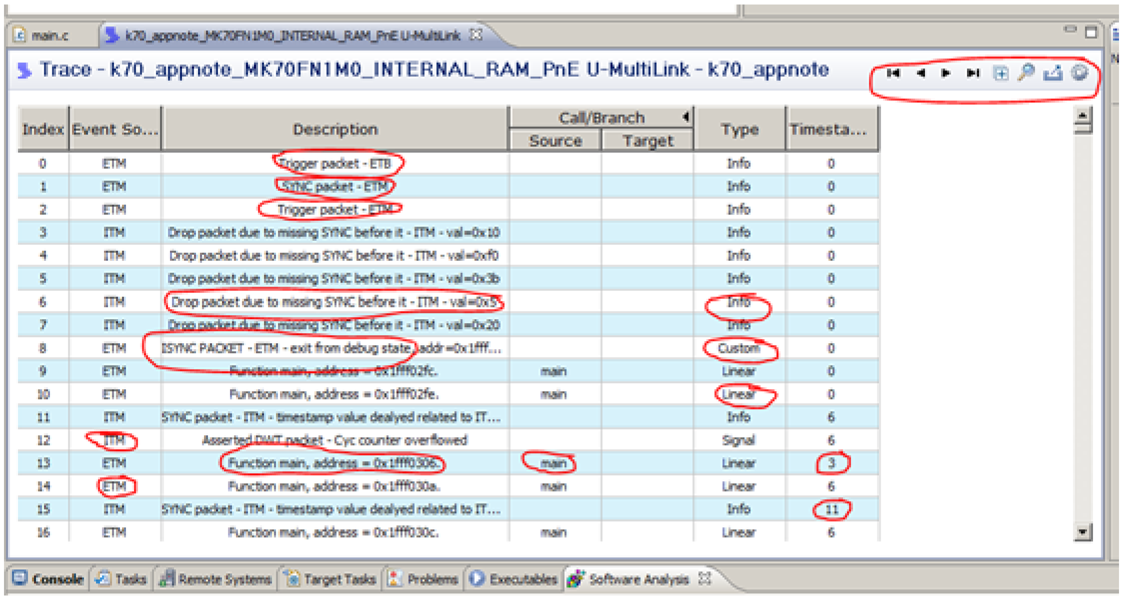


Figure 22. Save/Delete results

3.6 Trace view

Trace view shows the collected trace.



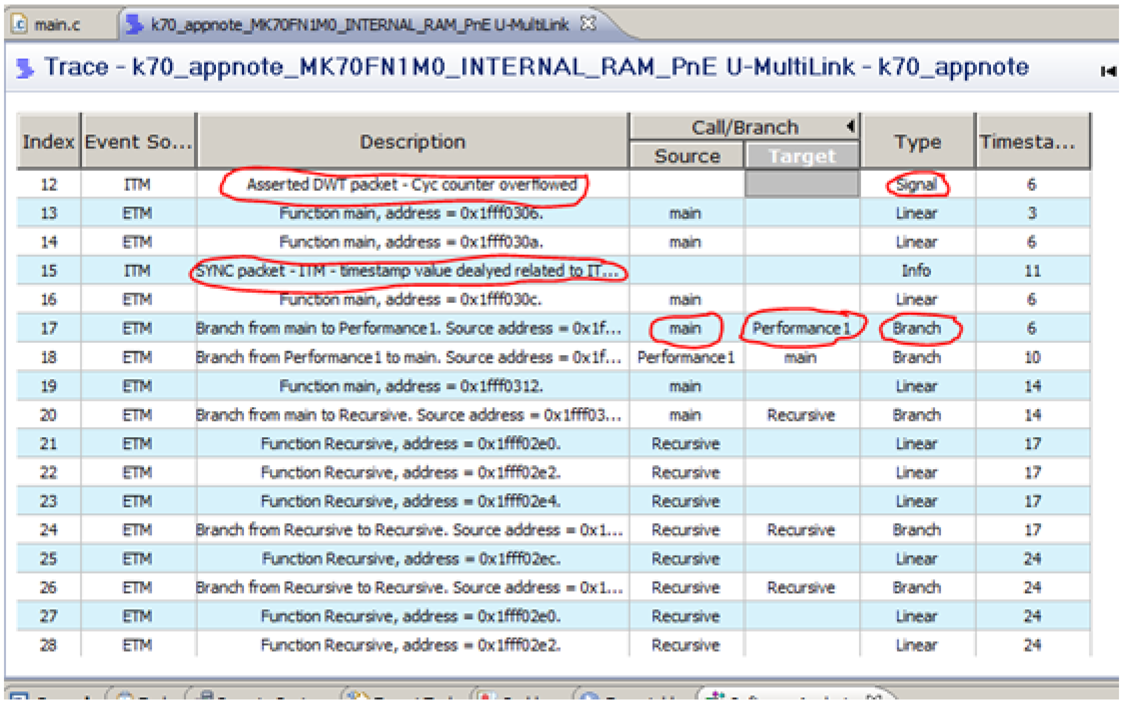
Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
0	ETM	Trigger packet - ETB			Info	0
1	ETM	SYNC packet - ETM			Info	0
2	ETM	Trigger packet - ETM			Info	0
3	ITM	Drop packet due to missing SYNC before it - ITM - val=0x10			Info	0
4	ITM	Drop packet due to missing SYNC before it - ITM - val=0xf0			Info	0
5	ITM	Drop packet due to missing SYNC before it - ITM - val=0x3b			Info	0
6	ITM	Drop packet due to missing SYNC before it - ITM - val=0x5			Info	0
7	ITM	Drop packet due to missing SYNC before it - ITM - val=0x20			Info	0
8	ETM	ISYNC PACKET - ETM - exit from debug state, addr=0x1fff...			Custom	0
9	ETM	Function main, address = 0x1fff02fc.	main		Linear	0
10	ETM	Function main, address = 0x1fff02fe.	main		Linear	0
11	ITM	SYNC packet - ITM - timestamp value dealyed related to IT...			Info	6
12	ITM	Asserted DWT packet - Cyc counter overflowed			Signal	6
13	ETM	Function main, address = 0x1fff0306.	main		Linear	3
14	ETM	Function main, address = 0x1fff030a.	main		Linear	6
15	ITM	SYNC packet - ITM - timestamp value dealyed related to IT...			Info	11
16	ETM	Function main, address = 0x1fff030c.	main		Linear	6

Figure 23. Trace viewer controls

Notice trigger packets (ETB and ETM), synchronization packets, drop packets (due to missed synchronization for ITM stream), function symbol name (for example, main), and timestamps (they are different for ITM and ETM events, because they use two different clock bases).

See the buttons on the upper-right side of the view, they allow to easily navigate in trace, expand trace events to see assembly and source code mixed in the same view, find a record, export to .csv format, and configure timestamp resolution.



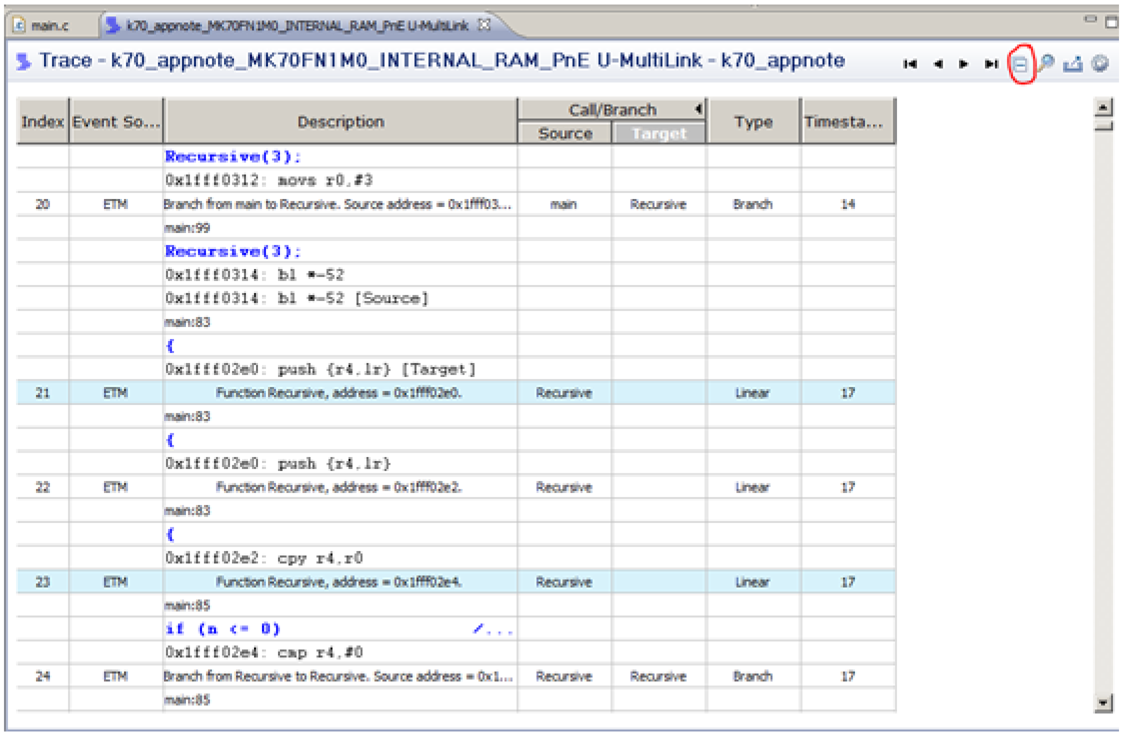
main.c | k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
12	ITM	Asserted DWT packet - Cyc counter overflowed			Signal	6
13	ETM	Function main, address = 0x1fff0306.	main		Linear	3
14	ETM	Function main, address = 0x1fff030a.	main		Linear	6
15	ITM	SYNC packet - ITM - timestamp value dealyed related to IT...			Info	11
16	ETM	Function main, address = 0x1fff030c.	main		Linear	6
17	ETM	Branch from main to Performance1. Source address = 0x1f...	main	Performance1	Branch	6
18	ETM	Branch from Performance1 to main. Source address = 0x1f...	Performance1	main	Branch	10
19	ETM	Function main, address = 0x1fff0312.	main		Linear	14
20	ETM	Branch from main to Recursive. Source address = 0x1fff03...	main	Recursive	Branch	14
21	ETM	Function Recursive, address = 0x1fff02e0.	Recursive		Linear	17
22	ETM	Function Recursive, address = 0x1fff02e2.	Recursive		Linear	17
23	ETM	Function Recursive, address = 0x1fff02e4.	Recursive		Linear	17
24	ETM	Branch from Recursive to Recursive. Source address = 0x1...	Recursive	Recursive	Branch	17
25	ETM	Function Recursive, address = 0x1fff02ec.	Recursive		Linear	24
26	ETM	Branch from Recursive to Recursive. Source address = 0x1...	Recursive	Recursive	Branch	24
27	ETM	Function Recursive, address = 0x1fff02e0.	Recursive		Linear	24
28	ETM	Function Recursive, address = 0x1fff02e2.	Recursive		Linear	24

Figure 24. Events in trace

Notice the event for a performance counter (Cycles counter overflowed) and also events for a branch, where both source and destination of the branch are displayed.



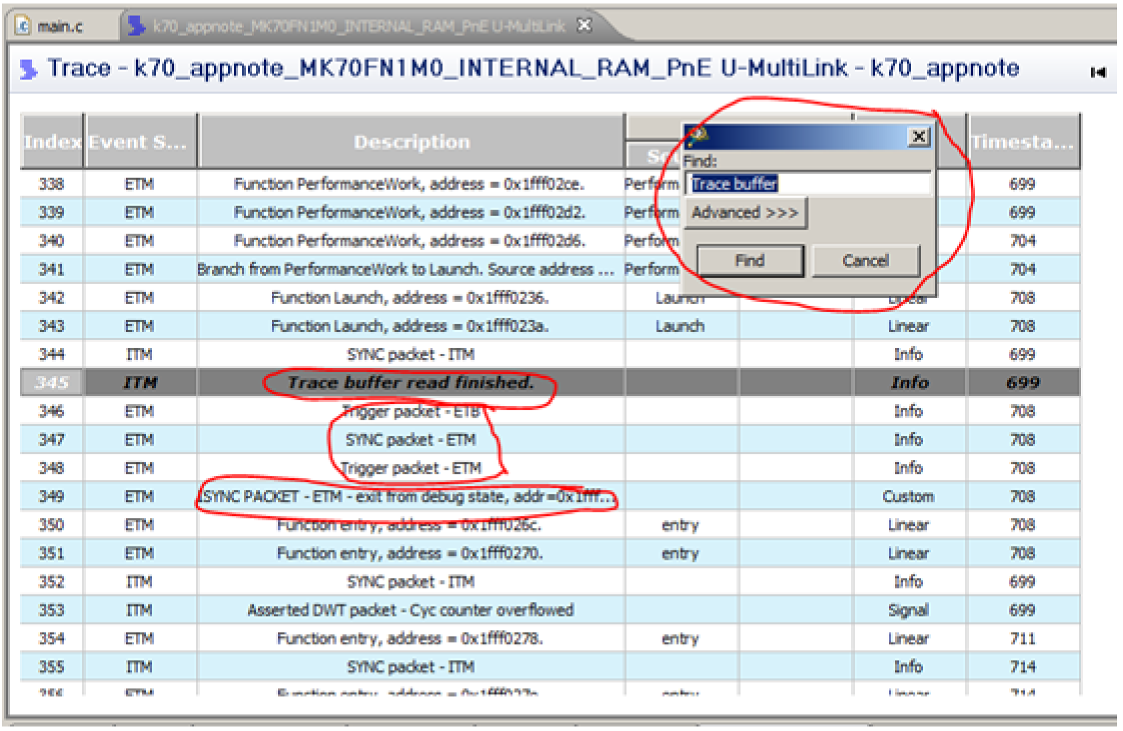
main.c | k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
		Recursive(3);				
		0x1fff0312: movs r0,#3				
20	ETM	Branch from main to Recursive. Source address = 0x1fff03...	main	Recursive	Branch	14
		main:99				
		Recursive(3);				
		0x1fff0314: bl *-52				
		0x1fff0314: bl *-52 [Source]				
		main:83				
		{				
		0x1fff02e0: push {r4,lr} [Target]				
21	ETM	Function Recursive, address = 0x1fff02e0.	Recursive		Linear	17
		main:83				
		{				
		0x1fff02e0: push {r4,lr}				
22	ETM	Function Recursive, address = 0x1fff02e2.	Recursive		Linear	17
		main:83				
		{				
		0x1fff02e2: cpy r4,r0				
23	ETM	Function Recursive, address = 0x1fff02e4.	Recursive		Linear	17
		main:85				
		if (n <= 0) /...				
		0x1fff02e4: cmp r4,#0				
24	ETM	Branch from Recursive to Recursive. Source address = 0x1...	Recursive	Recursive	Branch	17
		main:85				

Figure 25. Mixed trace (source and assembly code)

By pressing on the Expand all button, you can see assembly and source code mixed in trace.



Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event S...	Description	Se			Timesta...
338	ETM	Function PerformanceWork, address = 0x1fff02ce.	Perform			699
339	ETM	Function PerformanceWork, address = 0x1fff02d2.	Perform			699
340	ETM	Function PerformanceWork, address = 0x1fff02d6.	Perform			704
341	ETM	Branch from PerformanceWork to Launch. Source address ...	Perform			704
342	ETM	Function Launch, address = 0x1fff0236.	Launch		Linear	708
343	ETM	Function Launch, address = 0x1fff023a.	Launch		Linear	708
344	ITM	SYNC packet - ITM			Info	699
345	***ITM***	***Trace buffer read finished.***			***Info***	***699***
346	ETM	Trigger packet - ETB			Info	708
347	ETM	SYNC packet - ETM			Info	708
348	ETM	Trigger packet - ETM			Info	708
349	ETM	SYNC PACKET - ETM - exit from debug state, addr=0x1fff...			Custom	708
350	ETM	Function entry, address = 0x1fff026c.	entry		Linear	708
351	ETM	Function entry, address = 0x1fff0270.	entry		Linear	708
352	ITM	SYNC packet - ITM			Info	699
353	ITM	Asserted DWT packet - Cyc counter overflowed			Signal	699
354	ETM	Function entry, address = 0x1fff0278.	entry		Linear	711
355	ITM	SYNC packet - ITM			Info	714

Figure 26. Find in trace

You can search for an event in trace. In picture above, see the special event Trace buffer read finished. This appears only for continuous mode collected trace and signal when a trace is full and is uploaded on the host. Notice that after this event (which means entering in debug mode for the target), trace packets for resynchronization are sent by the ETM and ITM modules.

3.7 Timeline view

The timeline view shows a logic analyzer like view of an application execution.

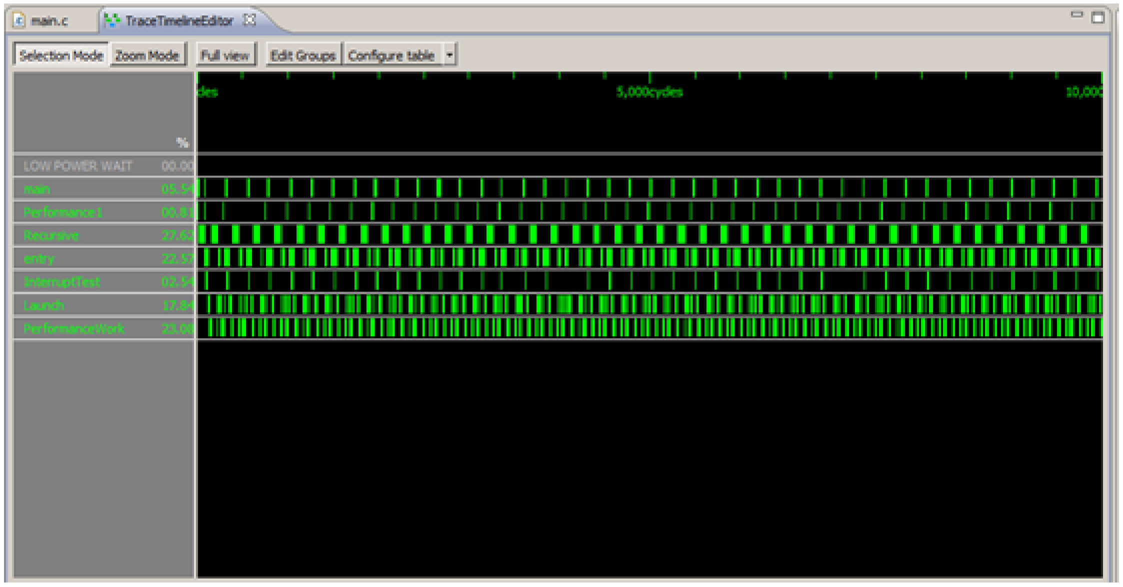


Figure 27. Timeline view

Notice that you have two cursors and you can make measurements on this view. On the left side, the executed function is shown together with the percentage from the execution time.

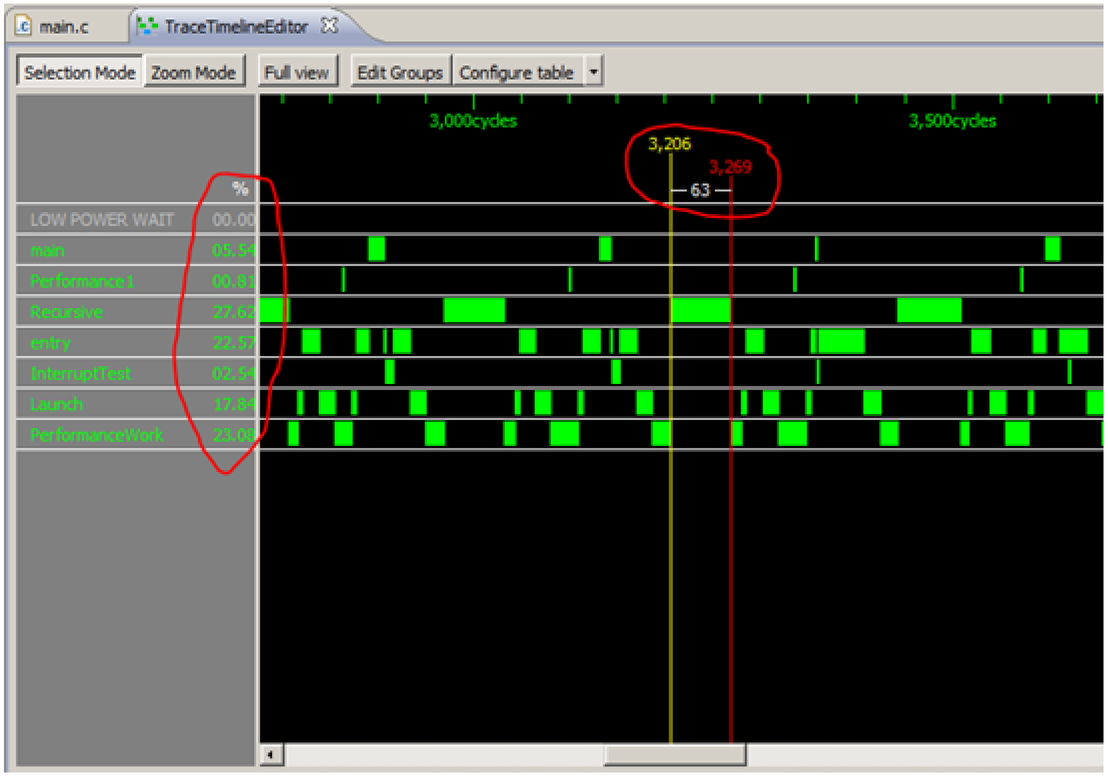
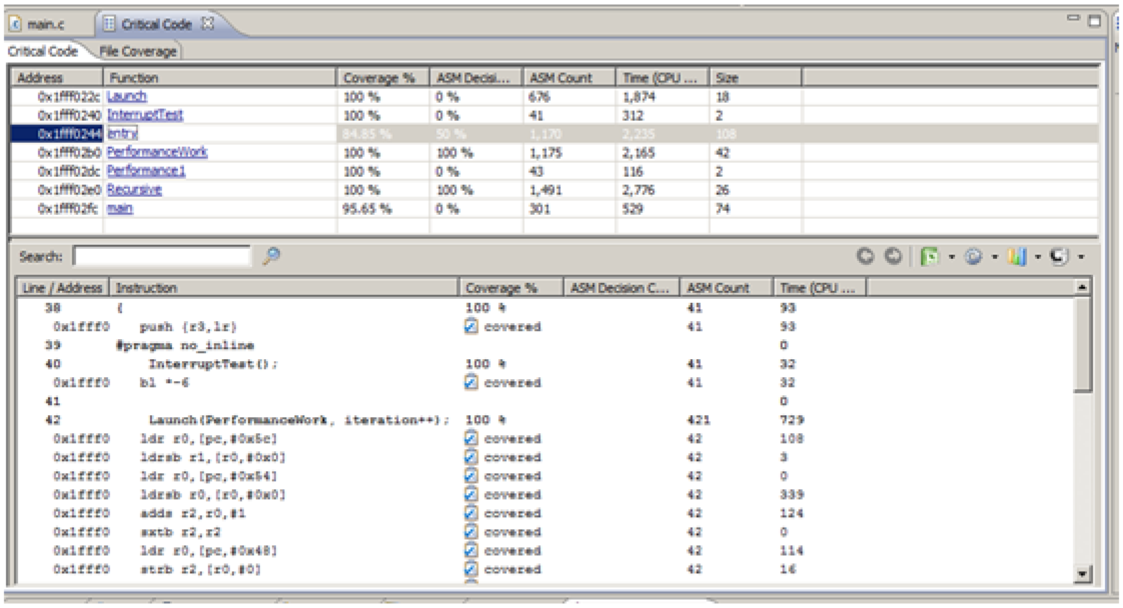


Figure 28. Timeline cursors

3.8 Critical code view

The critical code shows a flat profiler (a profiler that counts each function execution by summing the individual instruction time; this is different from an hierarchical profiler presented in next section) and code coverage.

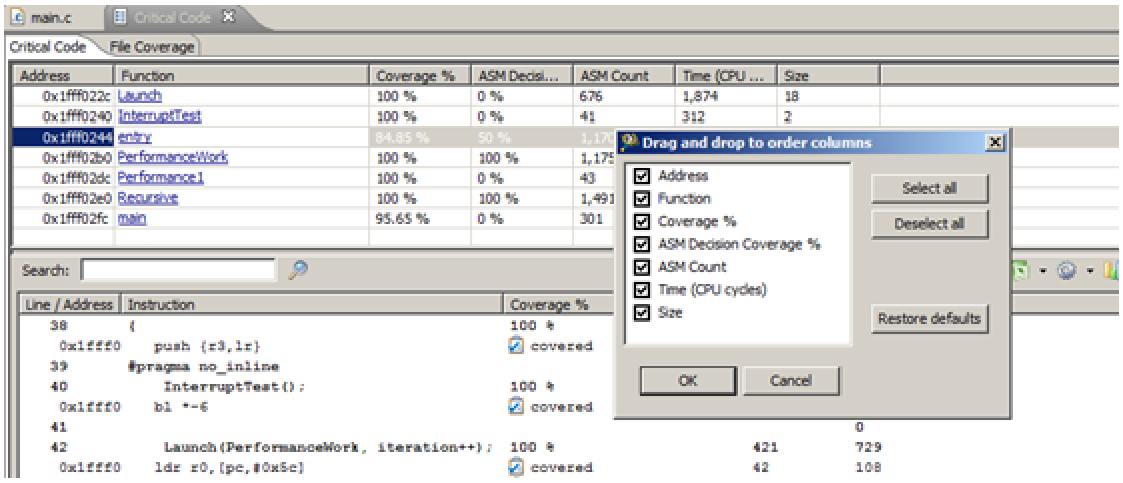


Address	Function	Coverage %	ASM Decisi...	ASM Count	Time (CPU ...	Size
0x1fff022c	Launch	100 %	0 %	676	1,874	18
0x1fff0240	InterruptTest	100 %	0 %	41	312	2
0x1fff0244	entry	84.85 %	50 %	1,170	2,235	108
0x1fff02b0	PerformanceWork	100 %	100 %	1,175	2,165	42
0x1fff02dc	Performance1	100 %	0 %	43	116	2
0x1fff02e0	Recursive	100 %	100 %	1,491	2,776	26
0x1fff02fc	main	95.65 %	0 %	301	529	74

Line / Address	Instruction	Coverage %	ASM Decision C...	ASM Count	Time (CPU ...
38	{	100 %		41	93
0x1fff0	push {r3,lr}	✓ covered		41	93
39	#pragma no_inline				0
40	InterruptTest();	100 %		41	32
0x1fff0	bl *-6	✓ covered		41	32
41					0
42	Launch(PerformanceWork, iteration++);	100 %		421	729
0x1fff0	ldr r0,[pc,#0x5c]	✓ covered		42	108
0x1fff0	ldrsb r1,[r0,#0x0]	✓ covered		42	3
0x1fff0	ldr r0,[pc,#0x54]	✓ covered		42	0
0x1fff0	ldrsb r0,[r0,#0x0]	✓ covered		42	339
0x1fff0	adds r2,r0,#1	✓ covered		42	124
0x1fff0	sxtb r2,r2	✓ covered		42	0
0x1fff0	ldr r0,[pc,#0x48]	✓ covered		42	114
0x1fff0	strb r2,[r0,#0]	✓ covered		42	16

Figure 29. Critical code view

You can configure what to see in the upper or bottom panels:



Drag and drop to order columns

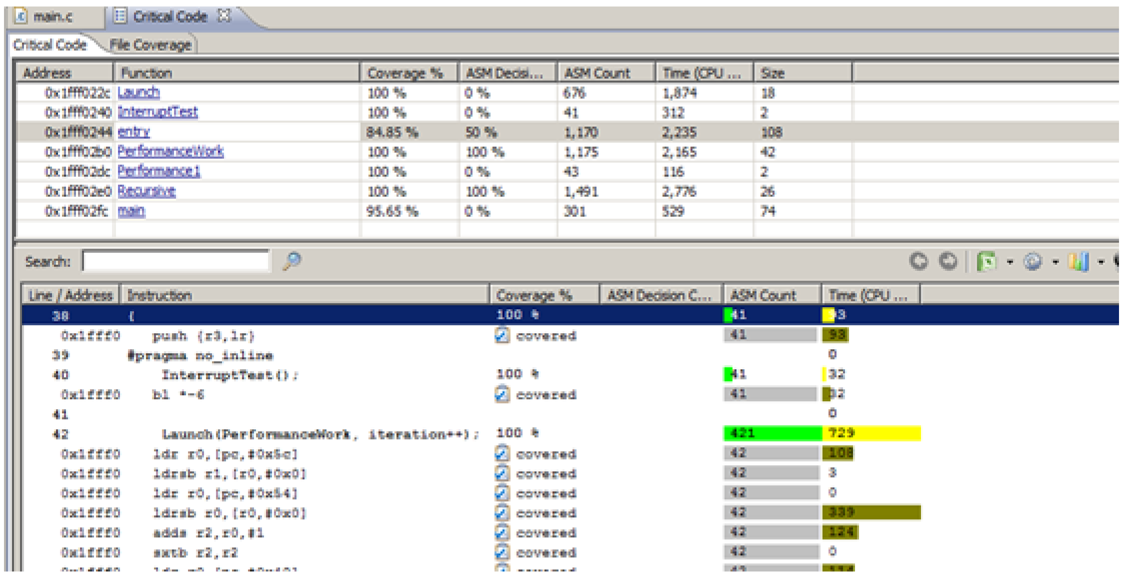
- ☑ Address
- ☑ Function
- ☑ Coverage %
- ☑ ASM Decision Coverage %
- ☑ ASM Count
- ☑ Time (CPU cycles)
- ☑ Size

Figure 30. Critical code view configuration

Notice the mixed view (assembly and source code) and statistics for every executed line. You can choose to express the values using colored bars to easily spot biggest time.

NOTE

Bar length is not scalable at the percentage of each execution. Instead, the biggest time has a bar showing 100% and the rest are relative. This is done to avoid little values to not be visible.



main.c | Critical Code

Critical Code | File Coverage

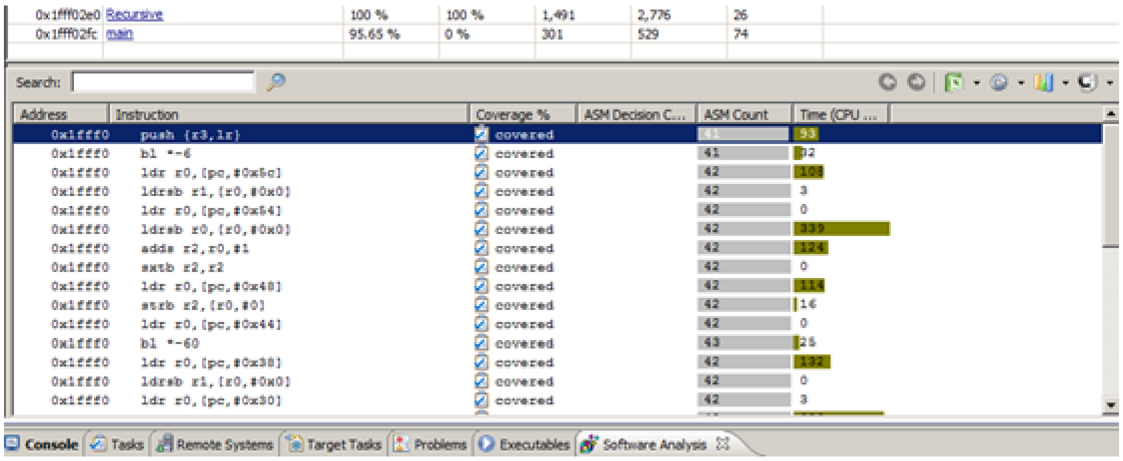
Address	Function	Coverage %	ASM Decisi...	ASM Count	Time (CPU ...	Size
0x1fff022c	Launch	100 %	0 %	676	1,874	18
0x1fff0240	InterruptTest	100 %	0 %	41	312	2
0x1fff0244	entry	84.85 %	50 %	1,170	2,235	108
0x1fff02b0	PerformanceWork	100 %	100 %	1,175	2,165	42
0x1fff02dc	Performance1	100 %	0 %	43	116	2
0x1fff02e0	Recursive	100 %	100 %	1,491	2,776	26
0x1fff02fc	main	95.65 %	0 %	301	529	74

Search:

Line / Address	Instruction	Coverage %	ASM Decision C...	ASM Count	Time (CPU ...
38	{	100 %		41	93
0x1fff0	push {r3,lr}	✓ covered		41	93
39	#pragma no_inline				0
40	InterruptTest();	100 %		41	32
0x1fff0	bl *-6	✓ covered		41	32
41					0
42	Launch(PerformanceWork, iteration++);	100 %		421	729
0x1fff0	ldr r0,[pc,#0x5c]	✓ covered		42	108
0x1fff0	ldrsb r1,[r0,#0x0]	✓ covered		42	3
0x1fff0	ldr r0,[pc,#0x54]	✓ covered		42	0
0x1fff0	ldrsb r0,[r0,#0x0]	✓ covered		42	339
0x1fff0	adds r2,r0,#1	✓ covered		42	124
0x1fff0	sxtb r2,r2	✓ covered		42	0

Figure 31. Critical code – mixed source code view

You have the option to show code only in assembly:



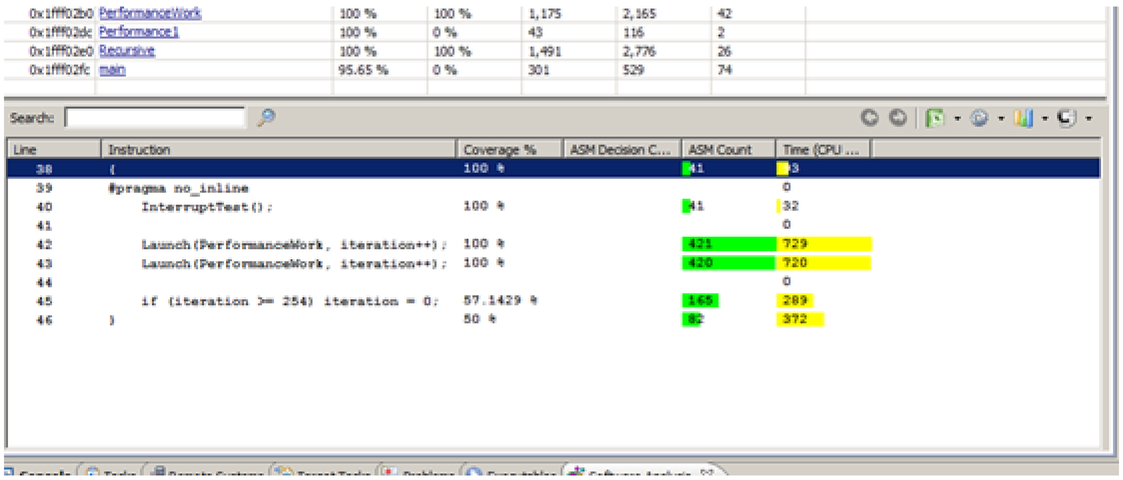
0x1fff02e0	Recursive	100 %	100 %	1,491	2,776	26
0x1fff02fc	main	95.65 %	0 %	301	529	74

Search:

Address	Instruction	Coverage %	ASM Decision C...	ASM Count	Time (CPU ...
0x1fff0	push {r3,lr}	✓ covered		41	93
0x1fff0	bl *-6	✓ covered		41	32
0x1fff0	ldr r0,[pc,#0x5c]	✓ covered		42	108
0x1fff0	ldrsb r1,[r0,#0x0]	✓ covered		42	3
0x1fff0	ldr r0,[pc,#0x54]	✓ covered		42	0
0x1fff0	ldrsb r0,[r0,#0x0]	✓ covered		42	339
0x1fff0	adds r2,r0,#1	✓ covered		42	124
0x1fff0	sxtb r2,r2	✓ covered		42	0
0x1fff0	ldr r0,[pc,#0x48]	✓ covered		42	114
0x1fff0	strb r2,[r0,#0]	✓ covered		42	16
0x1fff0	ldr r0,[pc,#0x44]	✓ covered		42	0
0x1fff0	bl *-60	✓ covered		43	25
0x1fff0	ldr r0,[pc,#0x38]	✓ covered		42	132
0x1fff0	ldrsb r1,[r0,#0x0]	✓ covered		42	0
0x1fff0	ldr r0,[pc,#0x30]	✓ covered		42	3

Figure 32. Critical code – assembly code view

Or only as C source code:



0x1fff02b0	PerformanceWork	100 %	100 %	1,175	2,165	42
0x1fff02dc	Performance1	100 %	0 %	43	116	2
0x1fff02e0	Recursive	100 %	100 %	1,491	2,776	26
0x1fff02fc	main	95.65 %	0 %	301	529	74

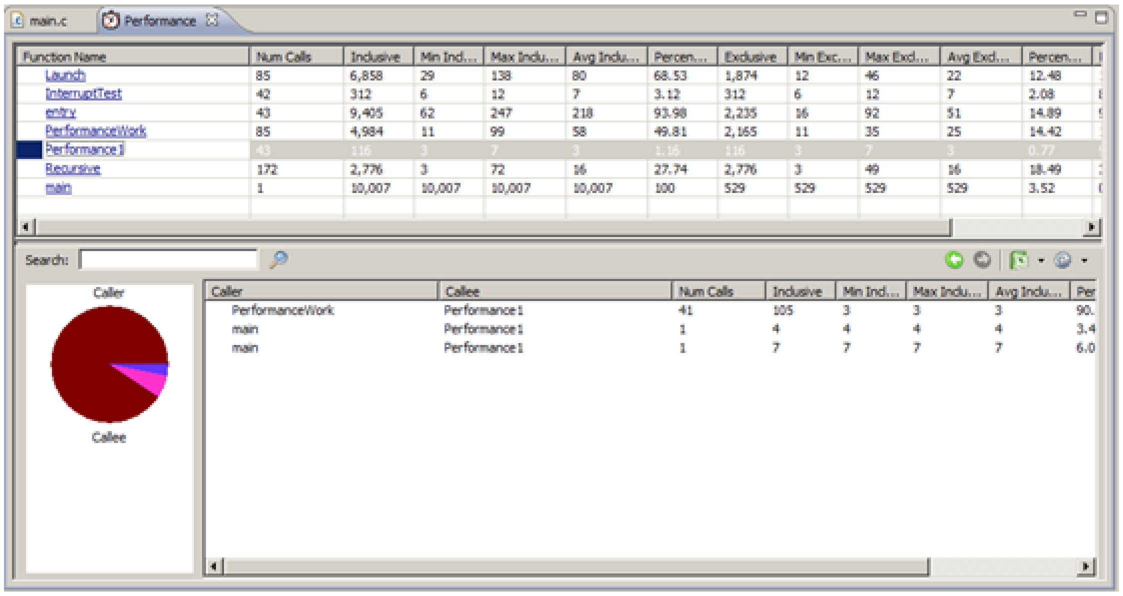
Search:

Line	Instruction	Coverage %	ASM Decision C...	ASM Count	Time (CPU ...
38	{	100 %		41	3
39	#pragma no_inline				0
40	InterruptTest();	100 %		41	32
41					0
42	Launch(PerformanceWork, iteration++);	100 %		421	729
43	Launch(PerformanceWork, iteration++);	100 %		420	720
44					0
45	if (iteration >= 254) iteration = 0;	57.1429 %		165	289
46	}	50 %		82	372

Figure 33. Critical code – source code view

3.9 Performance view

The performance view is a hierarchical profiler (it shows functions statistics calculated as parent-child or caller-calee pairs). It presents self-time (exclusive time) for a function (time spent only in that function) and hierarchical time (inclusive time) for a function (time spent in that function and in all its children functions).



main.c | Performance

Function Name	Num Calls	Inclusive	Min Ind...	Max Indu...	Avg Indu...	Percen...	Exclusive	Min Exc...	Max Excl...	Avg Excl...	Percen...
Launch	85	6,858	29	138	80	68.53	1,874	12	46	22	12.48
InterruptTest	42	312	6	12	7	3.12	312	6	12	7	2.08
entry	43	9,405	62	247	218	93.98	2,235	16	92	51	14.89
PerformanceWork	85	4,984	11	99	58	49.81	2,165	11	35	25	14.42
Performance1	43	116	3	7	3	1.16	116	3	7	3	0.77
Recursive	172	2,776	3	72	16	27.74	2,776	3	49	16	18.49
main	1	10,007	10,007	10,007	10,007	100	529	529	529	529	3.52

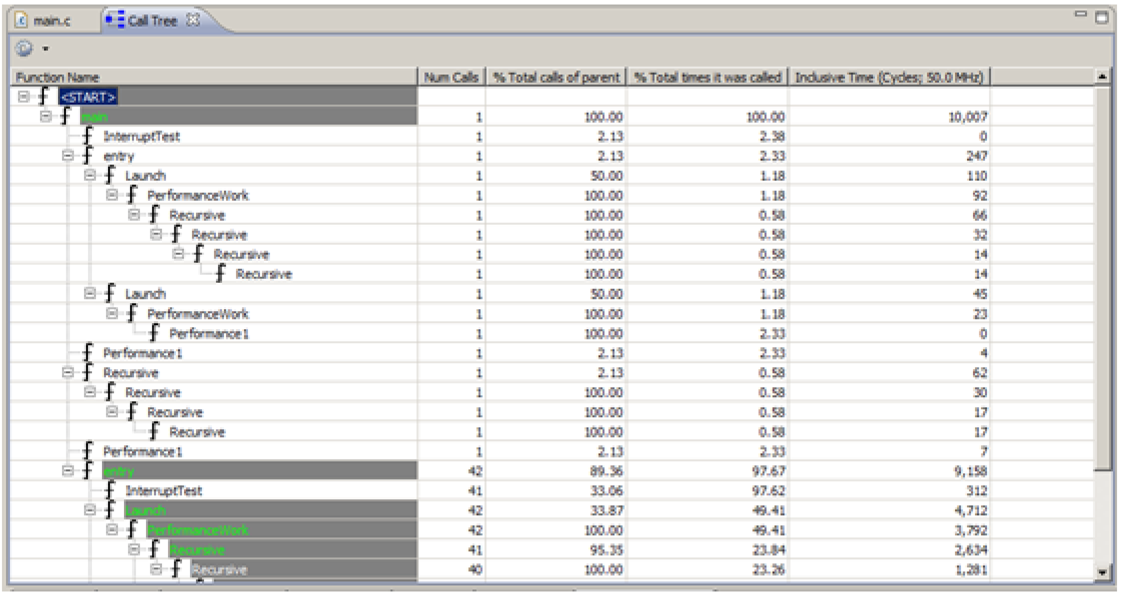
Search:

Caller	Callee	Num Calls	Inclusive	Min Ind...	Max Indu...	Avg Indu...	Per
PerformanceWork	Performance1	41	105	3	3	3	90.
main	Performance1	1	4	4	4	4	3.4
main	Performance1	1	7	7	7	7	6.0

Figure 34. Performance view

3.10 Call Tree view

The Call Tree view shows functions calling the tree together with some statistics.



main.c | Call Tree

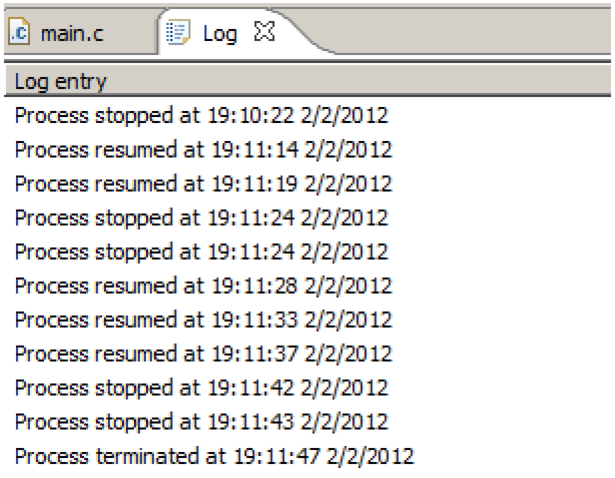
Function Name	Num Calls	% Total calls of parent	% Total times it was called	Inclusive Time (Cycles; 50.0 MHz)
<START>				
main	1	100.00	100.00	10,007
InterruptTest	1	2.13	2.38	0
entry	1	2.13	2.33	247
Launch	1	50.00	1.18	110
PerformanceWork	1	100.00	1.18	92
Recursive	1	100.00	0.58	66
Recursive	1	100.00	0.58	32
Recursive	1	100.00	0.58	14
Recursive	1	100.00	0.58	14
Launch	1	50.00	1.18	45
PerformanceWork	1	100.00	1.18	23
Performance1	1	100.00	2.33	0
Performance1	1	2.13	2.33	4
Recursive	1	2.13	0.58	62
Recursive	1	100.00	0.58	30
Recursive	1	100.00	0.58	17
Recursive	1	100.00	0.58	17
Performance1	1	2.13	2.33	7
entry	42	89.36	97.67	9,158
InterruptTest	41	33.06	97.62	312
Launch	42	33.87	49.41	4,712
PerformanceWork	42	100.00	49.41	3,792
Recursive	41	95.35	23.84	2,634
Recursive	40	100.00	23.26	1,281

Figure 35. Call Tree view

Notice the dark grey nodes; they indicate the critical call chain, this means the chain of functions that consume most of the time.

3.11 Log view

If it is enabled, the log will display the interaction of CodeWarrior with the target. You can see how intrusive was the collection of trace or you can analyze your activity (debugging) for an application execution.



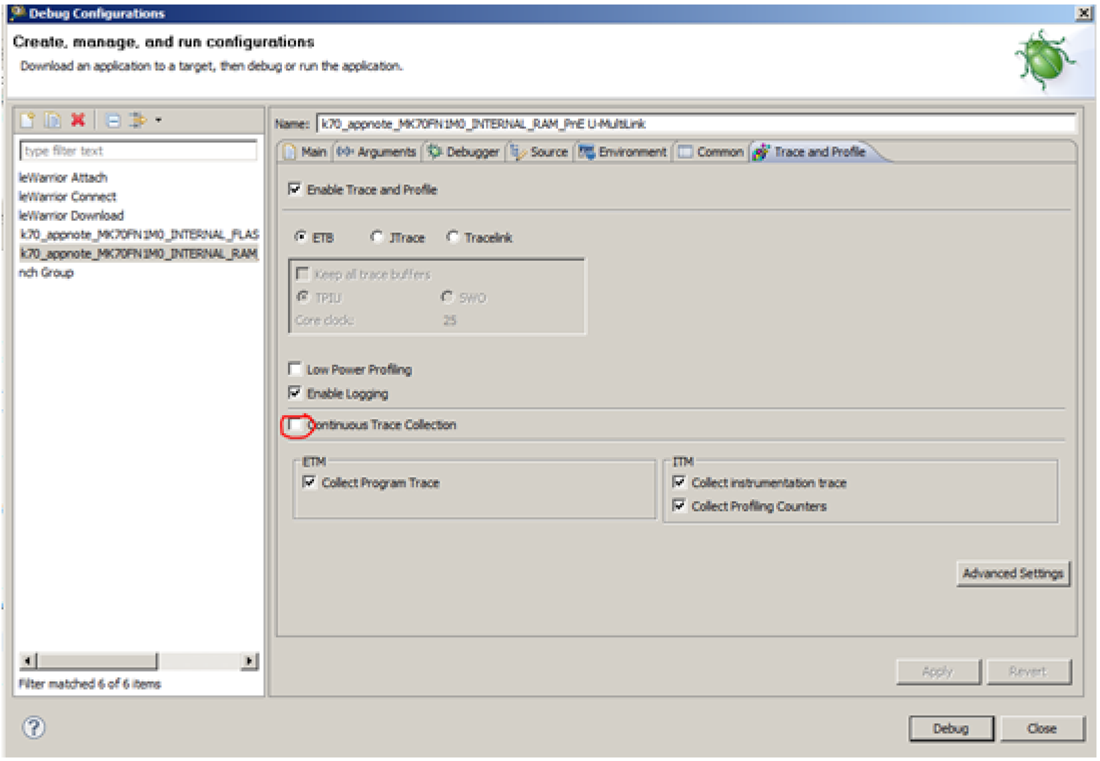
main.c | Log

```
Log entry
Process stopped at 19:10:22 2/2/2012
Process resumed at 19:11:14 2/2/2012
Process resumed at 19:11:19 2/2/2012
Process stopped at 19:11:24 2/2/2012
Process stopped at 19:11:24 2/2/2012
Process resumed at 19:11:28 2/2/2012
Process resumed at 19:11:33 2/2/2012
Process resumed at 19:11:37 2/2/2012
Process stopped at 19:11:42 2/2/2012
Process stopped at 19:11:43 2/2/2012
Process terminated at 19:11:47 2/2/2012
```

Figure 36. Log view

3.12 Trace in overwrite mode

The figure below shows how to configure trace collected in overwrite mode.


Debug Configurations
Create, manage, and run configurations
Download an application to a target, then debug or run the application.
type filter text
leWarrior Attach
leWarrior Connect
leWarrior Download
k70_appnote_MK70FN1M0_INTERNAL_FLAS
k70_appnote_MK70FN1M0_INTERNAL_RAM
nch Group
Filter matched 6 of 6 items
Main
Arguments
Debugger
Source
Environment
Common
Trace and Profile
Advanced Settings
Apply
Revert
Debug
Close

Name: k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

[✓] Enable Trace and Profile

[● ETB] ○ JTrace ○ Tracelink

[ ] Keep all trace buffers
[● TPIU] ○ SWO
Core clock: 25

[ ] Low Power Profiling
[✓] Enable Logging
[ ] Continuous Trace Collection

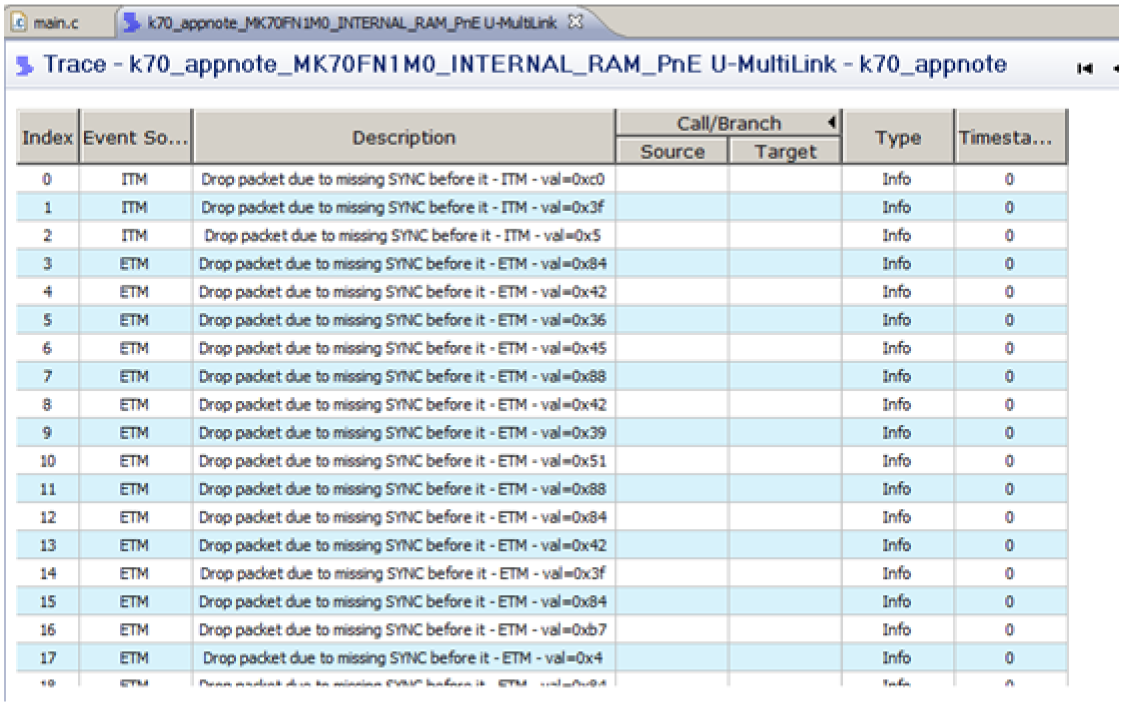
ETM
[✓] Collect Program Trace

ITM
[✓] Collect instrumentation trace
[✓] Collect Profiling Counters

Figure 37. Configure trace for overwrite mode

You need to make sure that Continuous Trace collection option is not checked. Overwrite collection mode is the default mode.

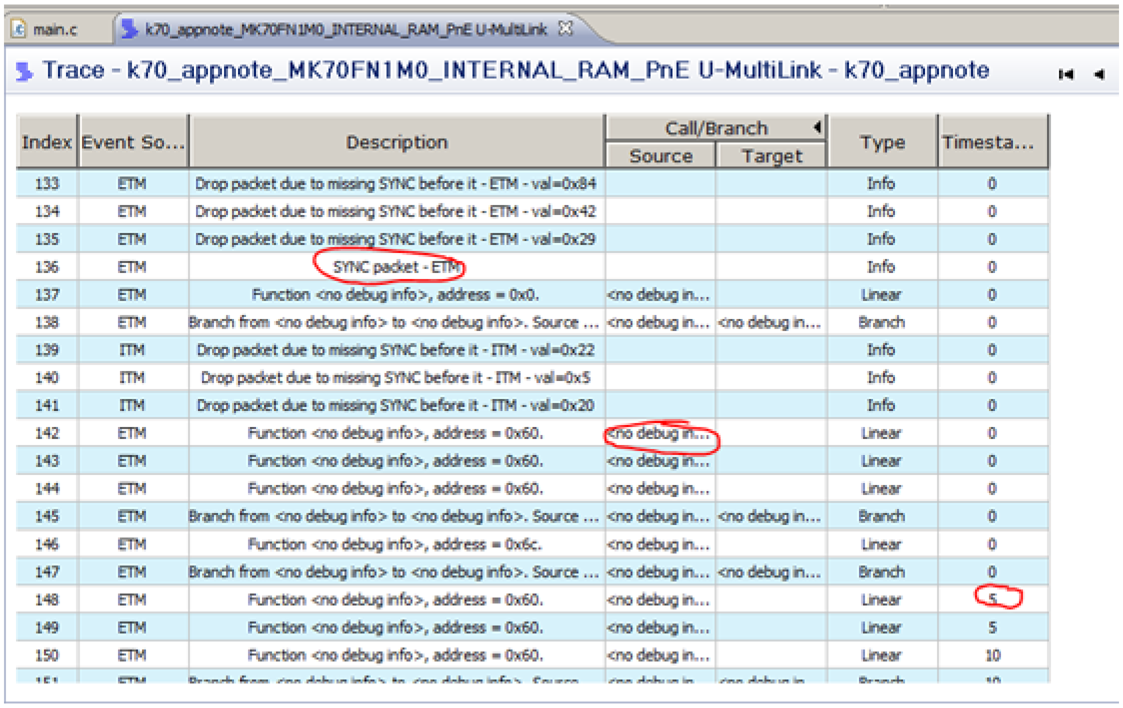
Please download the application and run it. Suspend it at some point, then Terminate application. Open Trace view.


main.c
k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink
Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
0	ITM	Drop packet due to missing SYNC before it - ITM - val=0xc0			Info	0
1	ITM	Drop packet due to missing SYNC before it - ITM - val=0x3f			Info	0
2	ITM	Drop packet due to missing SYNC before it - ITM - val=0x5			Info	0
3	ETM	Drop packet due to missing SYNC before it - ETM - val=0x84			Info	0
4	ETM	Drop packet due to missing SYNC before it - ETM - val=0x42			Info	0
5	ETM	Drop packet due to missing SYNC before it - ETM - val=0x36			Info	0
6	ETM	Drop packet due to missing SYNC before it - ETM - val=0x45			Info	0
7	ETM	Drop packet due to missing SYNC before it - ETM - val=0x88			Info	0
8	ETM	Drop packet due to missing SYNC before it - ETM - val=0x42			Info	0
9	ETM	Drop packet due to missing SYNC before it - ETM - val=0x39			Info	0
10	ETM	Drop packet due to missing SYNC before it - ETM - val=0x51			Info	0
11	ETM	Drop packet due to missing SYNC before it - ETM - val=0x88			Info	0
12	ETM	Drop packet due to missing SYNC before it - ETM - val=0x84			Info	0
13	ETM	Drop packet due to missing SYNC before it - ETM - val=0x42			Info	0
14	ETM	Drop packet due to missing SYNC before it - ETM - val=0x3f			Info	0
15	ETM	Drop packet due to missing SYNC before it - ETM - val=0x84			Info	0
16	ETM	Drop packet due to missing SYNC before it - ETM - val=0xb7			Info	0
17	ETM	Drop packet due to missing SYNC before it - ETM - val=0x4			Info	0
18	ETM	Drop packet due to missing SYNC before it - ETM - val=0x84			Info	0

Figure 38. Dropped packets in trace

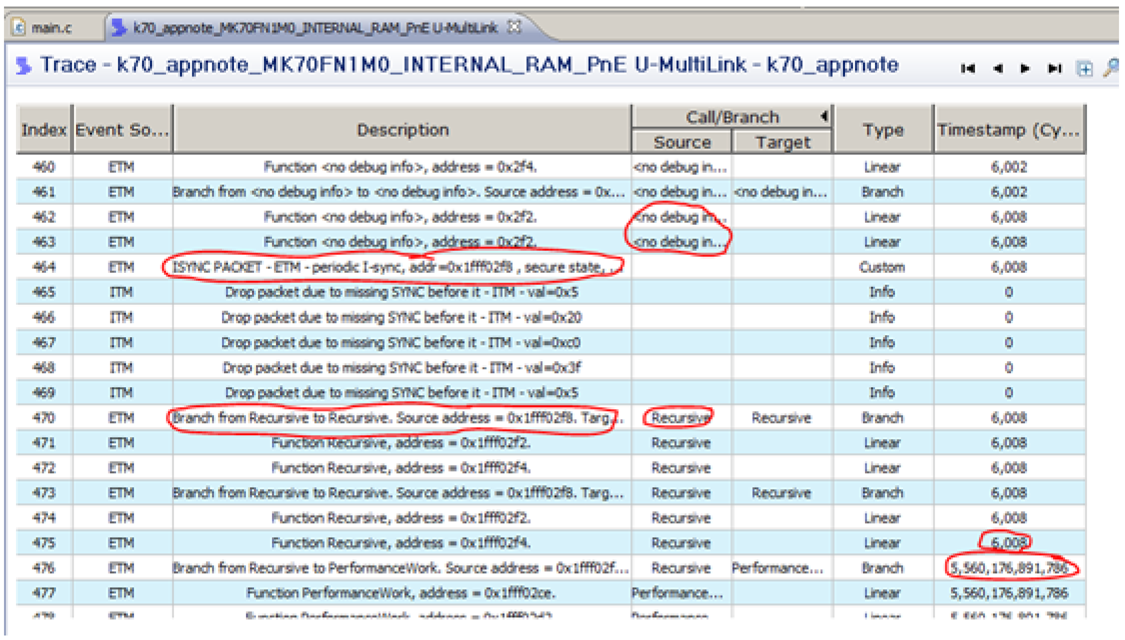
Notice that start of trace has a lot of dropped packets. This is because in overwrite mode, trace has big chances to overwrite the synchronization packets.


main.c
k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink
Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
133	ETM	Drop packet due to missing SYNC before it - ETM - val=0x84			Info	0
134	ETM	Drop packet due to missing SYNC before it - ETM - val=0x42			Info	0
135	ETM	Drop packet due to missing SYNC before it - ETM - val=0x29			Info	0
136	ETM	SYNC packet - ETM			Info	0
137	ETM	Function <no debug info>, address = 0x0.	<no debug in...		Linear	0
138	ETM	Branch from <no debug info> to <no debug info>. Source ...	<no debug in...	<no debug in...	Branch	0
139	ITM	Drop packet due to missing SYNC before it - ITM - val=0x22			Info	0
140	ITM	Drop packet due to missing SYNC before it - ITM - val=0x5			Info	0
141	ITM	Drop packet due to missing SYNC before it - ITM - val=0x20			Info	0
142	ETM	Function <no debug info>, address = 0x60.	<no debug in...		Linear	0
143	ETM	Function <no debug info>, address = 0x60.	<no debug in...		Linear	0
144	ETM	Function <no debug info>, address = 0x60.	<no debug in...		Linear	0
145	ETM	Branch from <no debug info> to <no debug info>. Source ...	<no debug in...	<no debug in...	Branch	0
146	ETM	Function <no debug info>, address = 0x6c.	<no debug in...		Linear	0
147	ETM	Branch from <no debug info> to <no debug info>. Source ...	<no debug in...	<no debug in...	Branch	0
148	ETM	Function <no debug info>, address = 0x60.	<no debug in...		Linear	5
149	ETM	Function <no debug info>, address = 0x60.	<no debug in...		Linear	5
150	ETM	Function <no debug info>, address = 0x60.	<no debug in...		Linear	10
151	ETM	Branch from <no debug info> to <no debug info>. Source ...	<no debug in...	<no debug in...	Branch	10

Figure 39. Partial trace synchronization in overwrite mode

See above how an ETM synchronization packet obligates the decoder to correctly recognize the ETM packets. However, notice that logic function is not yet restored, since at this point addresses and time is relative to 0. An address and a timestamp syncronization packet is needed to correctly decode the stream.



main.c | k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timestamp (Cy...
			Source	Target		
460	ETM	Function <no debug info>, address = 0x2f4.	<no debug in...		Linear	6,002
461	ETM	Branch from <no debug info> to <no debug info>. Source address = 0x...	<no debug in...	<no debug in...	Branch	6,002
462	ETM	Function <no debug info>, address = 0x2f2.	<no debug in...		Linear	6,008
463	ETM	Function <no debug info>, address = 0x2f2.	<no debug in...		Linear	6,008
464	ETM	ISYNC PACKET - ETM - periodic I-sync, addr =0x1fff02f8 , secure state, ...			Custom	6,008
465	ITM	Drop packet due to missing SYNC before it - ITM - val=0x5			Info	0
466	ITM	Drop packet due to missing SYNC before it - ITM - val=0x20			Info	0
467	ITM	Drop packet due to missing SYNC before it - ITM - val=0xc0			Info	0
468	ITM	Drop packet due to missing SYNC before it - ITM - val=0x3f			Info	0
469	ITM	Drop packet due to missing SYNC before it - ITM - val=0x5			Info	0
470	ETM	Branch from Recursive to Recursive. Source address = 0x1fff02f8. Targ...	Recursive	Recursive	Branch	6,008
471	ETM	Function Recursive, address = 0x1fff02f2.	Recursive		Linear	6,008
472	ETM	Function Recursive, address = 0x1fff02f4.	Recursive		Linear	6,008
473	ETM	Branch from Recursive to Recursive. Source address = 0x1fff02f8. Targ...	Recursive	Recursive	Branch	6,008
474	ETM	Function Recursive, address = 0x1fff02f2.	Recursive		Linear	6,008
475	ETM	Function Recursive, address = 0x1fff02f4.	Recursive		Linear	6,008
476	ETM	Branch from Recursive to PerformanceWork. Source address = 0x1fff02f...	Recursive	Performance...	Branch	5,560,176,891,786
477	ETM	Function PerformanceWork, address = 0x1fff02ce.	Performance...		Linear	5,560,176,891,786

Figure 40. Complete trace synchronization in overwrite mode

See above how a synchronization packet for address makes trace to correctly decode the functions and later, timestamp sync packets (not specifically shown in trace) display the correct timestamp (see the jump from 6,008 to 5,560,176,891,786 cycles). From this point, the trace is perfectly synchronized, both for address and timestamp.

3.13 Trace collected with an external device

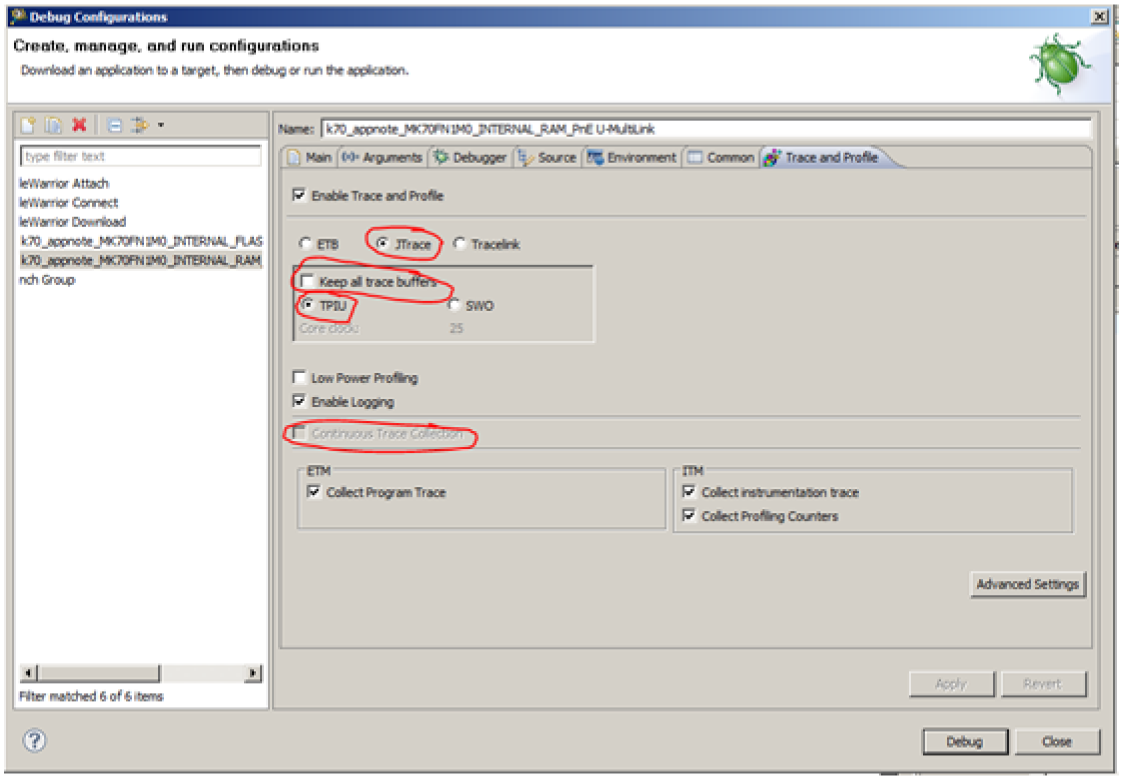
An external device (Segger J-Trace or P&E TraceLink) can be used to collect trace in overwrite mode, but with a bigger buffer (internal buffer ETB has 2 KB, external device J-Trace has 8 MB, external device TraceLink has 128 MB).

NOTE

The internal bus from ETM and ITM to Trace funnel (common sink, that mixes the two trace streams in one stream) is 8-bits wide. The path from trace funnel to ETB is on 32 bits, so it allows to buffer packets that come at great speed. That's why, ETB has the lowest rate of lost trace.

From Trace funnel to TPIU, the bus is on 8 bits. From TPIU to TPA (Trace Port Adapter) where external devices are plugged, the bus can be set to 1-bit, 2-bits or 4-bits. Notice that this is a bottle-neck and causes losing trace when it comes with a high frequency rate.

This is how to set trace collection to use the Segger J-Trace external device:



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

leWarrior Attach
leWarrior Connect
leWarrior Download
k70_appnote_MK70FN1M0_INTERNAL_FLAS
k70_appnote_MK70FN1M0_INTERNAL_RAM
nch Group

Filter matched 6 of 6 items

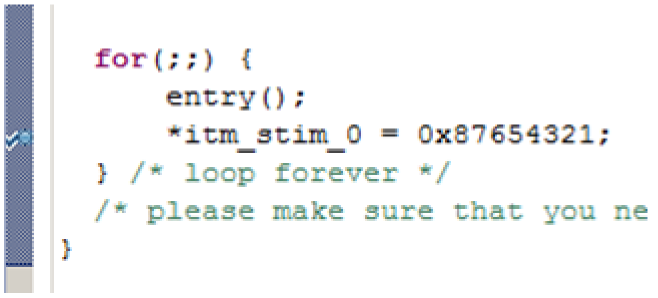
Name: k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

- [✓] Enable Trace and Profile
- ETB [✓ JTrace] Tracelink
- [ ] Keep all trace buffers
- [✓ TPIU] SWO
- Core clock: 25
- [ ] Low Power Profiling
- [✓] Enable Logging
- [ ] Continuous Trace Collection
- ETM: [✓] Collect Program Trace
- ITM: [✓] Collect instrumentation trace, [✓] Collect Profiling Counters

Figure 41. Configure trace collection with Segger J-Trace

With J-Trace, Continuous Trace Collection is grayed, because only the overwrite mode can be performed. However, you have Keep all trace buffer option that enables the Pseudo-continuous collection mode. In this mode, whenever suspend application execution trace is collected and kept in CodeWarrior. At next suspend, the new trace is appended to the old one.

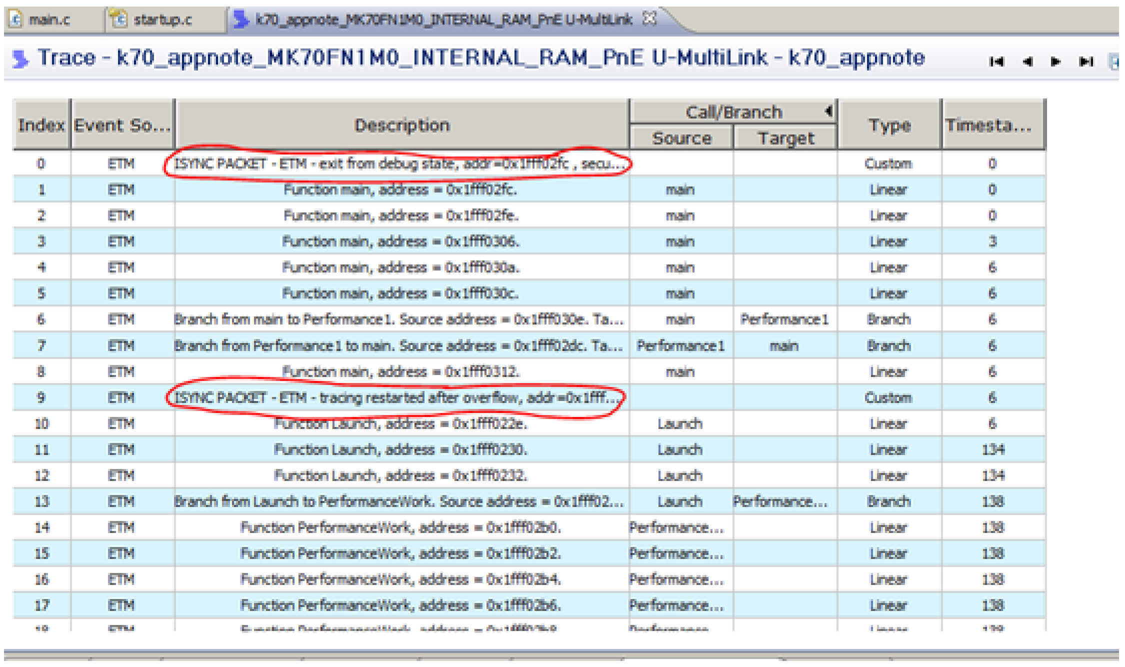
Download application on target and set a breakpoint as shown below. We do this to not collect too much trace, because decoding a large trace takes some time. To demonstrate Software Analsysis basic concepts it is ok to collect a small amount of trace.



```
    for(;;) {
        entry();
        *itm_stim_0 = 0x87654321;
    } /* loop forever */
    /* please make sure that you ne
}
```

Figure 42. Setting a breakpoint

Resume application and when it stops in breakpoint, Terminate application. Open Trace view.



main.c | startup.c | k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
0	ETM	ISYNC PACKET - ETM - exit from debug state, addr=0x1fff02fc , secu...			Custom	0
1	ETM	Function main, address = 0x1fff02fc.	main		Linear	0
2	ETM	Function main, address = 0x1fff02fe.	main		Linear	0
3	ETM	Function main, address = 0x1fff0306.	main		Linear	3
4	ETM	Function main, address = 0x1fff030a.	main		Linear	6
5	ETM	Function main, address = 0x1fff030c.	main		Linear	6
6	ETM	Branch from main to Performance1. Source address = 0x1fff030e. Ta...	main	Performance1	Branch	6
7	ETM	Branch from Performance1 to main. Source address = 0x1fff02dc. Ta...	Performance1	main	Branch	6
8	ETM	Function main, address = 0x1fff0312.	main		Linear	6
9	ETM	ISYNC PACKET - ETM - tracing restarted after overflow, addr=0x1fff...			Custom	6
10	ETM	Function Launch, address = 0x1fff022e.	Launch		Linear	6
11	ETM	Function Launch, address = 0x1fff0230.	Launch		Linear	134
12	ETM	Function Launch, address = 0x1fff0232.	Launch		Linear	134
13	ETM	Branch from Launch to PerformanceWork. Source address = 0x1fff02...	Launch	Performance...	Branch	138
14	ETM	Function PerformanceWork, address = 0x1fff02b0.	Performance...		Linear	138
15	ETM	Function PerformanceWork, address = 0x1fff02b2.	Performance...		Linear	138
16	ETM	Function PerformanceWork, address = 0x1fff02b4.	Performance...		Linear	138
17	ETM	Function PerformanceWork, address = 0x1fff02b6.	Performance...		Linear	138

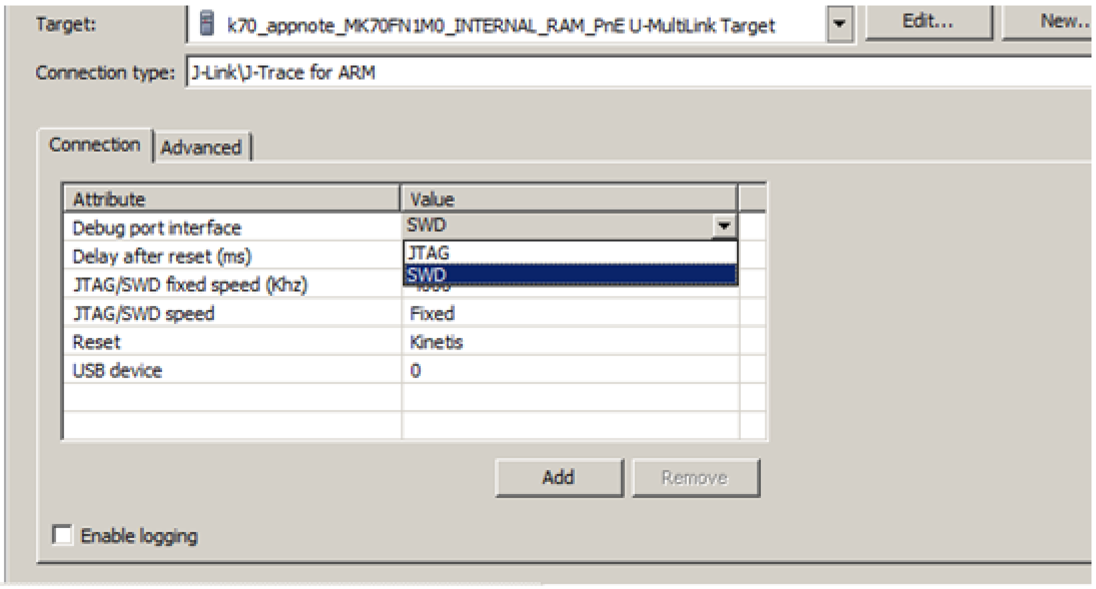
Figure 43. Trace overflow

Notice that the trace starts from the first record, there are no more triggers in trace. Also remark the synchronization packet that is put by trace after a trace lost (tracing restarted after overflow). This overflow is ETM internal overflow, because the external device cannot read the whole trace at the same rate it is generated.

3.14 SWO trace

The SWO trace is a light trace. Only ITM trace is recorded this way. SWO requires the debug to be executed over SWD interface. In this paper, a Segger J-Trace external device was used to collect SWO trace.

See below how to change the debugger option from using JTAG to use SWD.



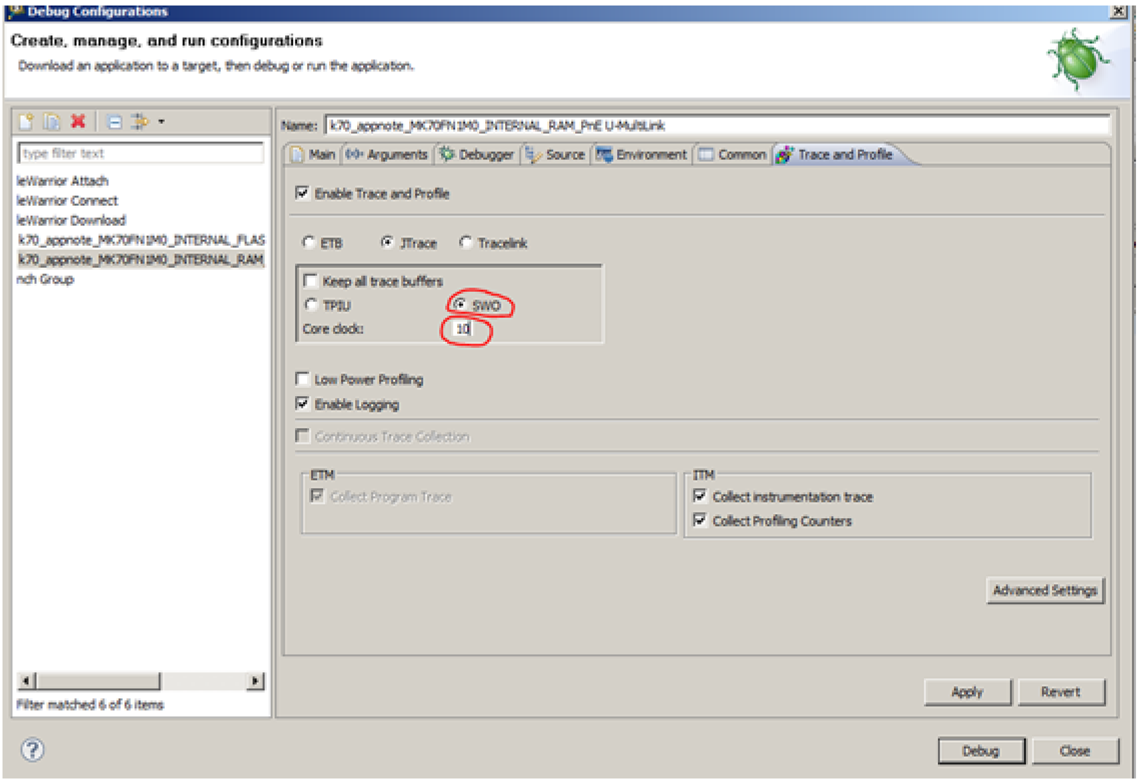
Target: k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink Target

Connection type: J-Link\J-Trace for ARM

Attribute	Value
Debug port interface	SWD
Delay after reset (ms)	
JTAG/SWD fixed speed (Khz)	
JTAG/SWD speed	Fixed
Reset	Kinetis
USB device	0

Figure 44. Configure debug connection for SWD

Open Trace and Profile settings and configure SWO clock.


Debug Configurations
Create, manage, and run configurations
Download an application to a target, then debug or run the application.
type filter text
leWarrior Attach
leWarrior Connect
leWarrior Download
k70_appnote_MK70FN1M0_INTERNAL_FLAS
k70_appnote_MK70FN1M0_INTERNAL_RAM
nch Group
Filter matched 6 of 6 items
Main
Arguments
Debugger
Source
Environment
Common
Trace and Profile
Advanced Settings
Apply
Revert
Debug
Close

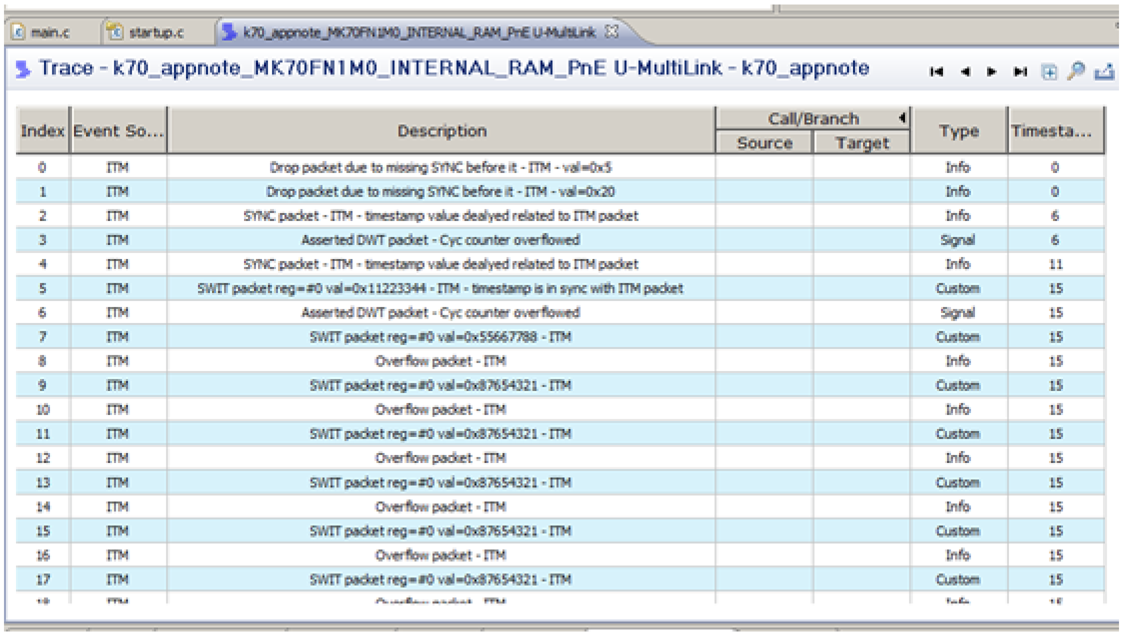
Name: k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

- [✓] Enable Trace and Profile
- ETB / [✓ JTrace] / Tracelink
- [ ] Keep all trace buffers
- TPIU / [✓ SWO]
- Core clock: 10
- [ ] Low Power Profiling
- [✓] Enable Logging
- [ ] Continuous Trace Collection
- ETM: [✓] Collect Program Trace
- ITM: [✓] Collect instrumentation trace
- ITM: [✓] Collect Profiling Counters

Figure 45. Configure trace collection with SWO

Notice that you need to provide a core clock to be used by SWO protocol. This is not the speed of the core, but it is somehow derived from it. For a K70 device (MK70FN1M0) this speed is 10 MHz.

Download the application on target and resume execution. Suspend application and notice that the trace is uploaded and decoded.



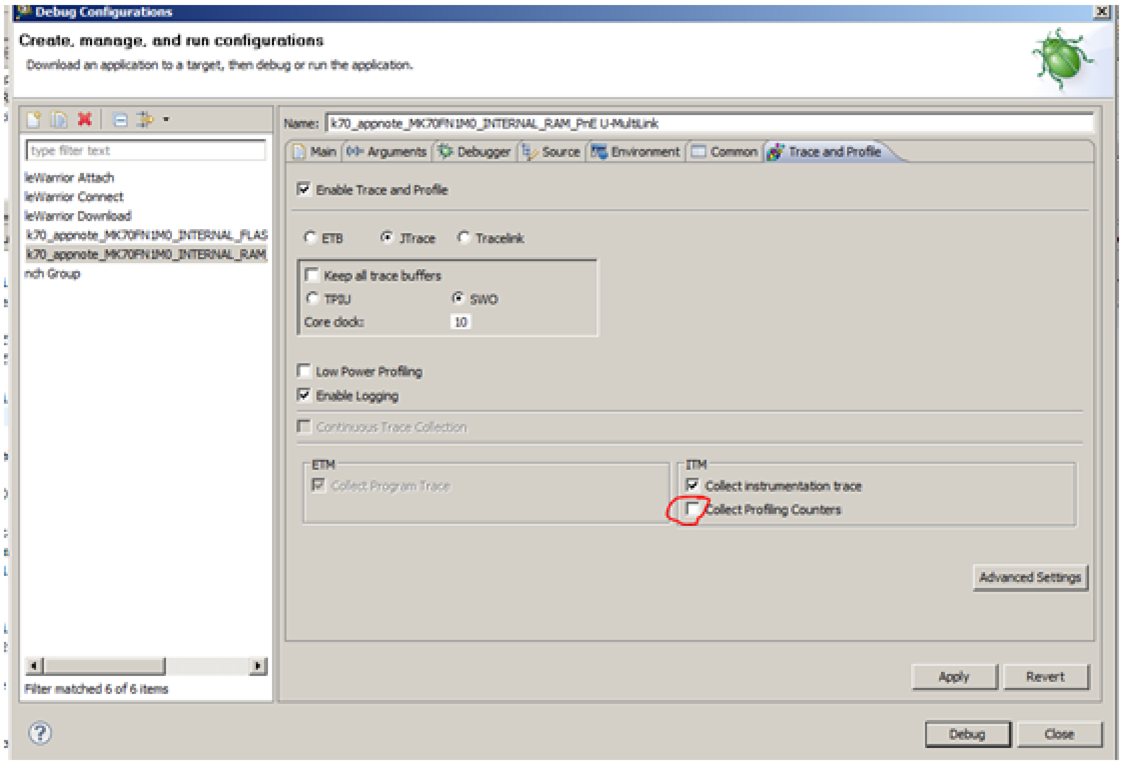
Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
0	ITM	Drop packet due to missing SYNC before it - ITM - val=0x5			Info	0
1	ITM	Drop packet due to missing SYNC before it - ITM - val=0x20			Info	0
2	ITM	SYNC packet - ITM - timestamp value dealyed related to ITM packet			Info	6
3	ITM	Asserted DWT packet - Cyc counter overflowed			Signal	6
4	ITM	SYNC packet - ITM - timestamp value dealyed related to ITM packet			Info	11
5	ITM	SWIT packet reg=#0 val=0x11223344 - ITM - timestamp is in sync with ITM packet			Custom	15
6	ITM	Asserted DWT packet - Cyc counter overflowed			Signal	15
7	ITM	SWIT packet reg=#0 val=0x55667788 - ITM			Custom	15
8	ITM	Overflow packet - ITM			Info	15
9	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	15
10	ITM	Overflow packet - ITM			Info	15
11	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	15
12	ITM	Overflow packet - ITM			Info	15
13	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	15
14	ITM	Overflow packet - ITM			Info	15
15	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	15
16	ITM	Overflow packet - ITM			Info	15
17	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	15

Figure 46. SWO trace

Notice that the values write from your code (by instrumentation of the code) are in trace (for example, 0x11223344, 0x87654321).

You can choose to not trace for Performance counters to free some bandwidth.



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

Name: k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink

[✓] Enable Trace and Profile

ETB [✓ JTrace] Tracelink

[ ] Keep all trace buffers

TPIU [✓ SWO]

Core clock: 10

[ ] Low Power Profiling

[✓] Enable Logging

[ ] Continuous Trace Collection

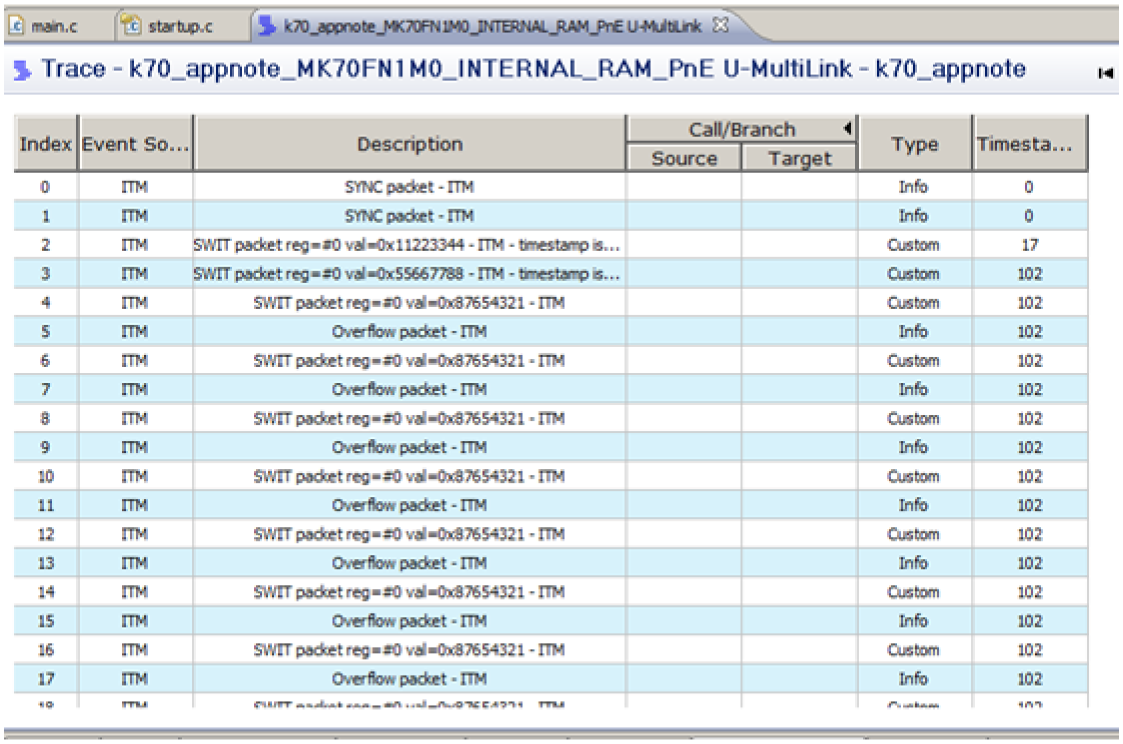
ETM: [✓] Collect Program Trace

ITM: [✓] Collect instrumentation trace

[ ] Collect Profiling Counters

Figure 47. Configure collection without Profiling Counters

The trace contains only instrumentation packets.



Trace - k70_appnote_MK70FN1M0_INTERNAL_RAM_PnE U-MultiLink - k70_appnote

Index	Event So...	Description	Call/Branch		Type	Timesta...
			Source	Target		
0	ITM	SYNC packet - ITM			Info	0
1	ITM	SYNC packet - ITM			Info	0
2	ITM	SWIT packet reg=#0 val=0x11223344 - ITM - timestamp is...			Custom	17
3	ITM	SWIT packet reg=#0 val=0x55667788 - ITM - timestamp is...			Custom	102
4	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	102
5	ITM	Overflow packet - ITM			Info	102
6	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	102
7	ITM	Overflow packet - ITM			Info	102
8	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	102
9	ITM	Overflow packet - ITM			Info	102
10	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	102
11	ITM	Overflow packet - ITM			Info	102
12	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	102
13	ITM	Overflow packet - ITM			Info	102
14	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	102
15	ITM	Overflow packet - ITM			Info	102
16	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	102
17	ITM	Overflow packet - ITM			Info	102
18	ITM	SWIT packet reg=#0 val=0x87654321 - ITM			Custom	102

Figure 48. SWO trace without Profiling Counters

4 Conclusions

Using the Software Analysis feature from CodeWarrior will help to better understand your application, to identify root causes for various issues (for example, application crash, and exceptions) and to optimize your application. There are plenty of features and ways to investigate your application. Choose your best fitted Software Analysis feature for your project and for your goal.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4506
Rev. 0, 6/2012