MCUXpresso IDE Linkserver Instruction Trace Guide

User guide

6 July, 2017

Copyright © 2017 NXP Semiconductors

1. Trace Overview

There are two different kinds of tracing technologies available directly within MCUXpresso IDE.

- **Instruction Trace** - capturing an instruction stream within onboard RAM of the MCU. This data can then be retrieved, decoded, and displayed within the IDE.
  - Available with any LinkServer (including CMSIS-DAP) compatible debug probes
- **SWO Trace** - capturing events occurring on a running MCU in real time
  - Only available with LPC-Link2 debug probes (using MCUXpresso IDE's CMSIS-DAP probe firmware)

The Trace functionality available depends on the features supported by your target MCU and the features supported by your debug probe.

Compatibility

MCUXpresso IDE's Instruction Trace functionality can only be used with LinkServer debug probes (i.e. it cannot be used with P&E or SEGGER J-Link debug probes). However, non-LinkServer probes may provide trace capabilities via other software – check the vendors' websites for more details.

The rest of this guide looks at Instruction Trace. For information regarding SWO Trace, see the SWO Trace guide.

1.1 Instruction Trace Overview

Instruction Trace provides the ability to record and review the sequence of instructions executed on a target. The MCUXpresso IDE provides support for Instruction Trace via on-chip RAM trace buffers. Instruction Trace makes use of the *Embedded Trace Buffer* (ETB) on Cortex M3 and M4 parts and the *Micro Trace Buffer* (MTB) on the Cortex-M0+. The instruction trace information, which is generated at high speed within the CPU, can be captured in real time and stored in these on-chip buffers, so that they can be downloaded at lower speeds without the need for an expensive external trace capture box.

The MCUXpresso IDE exposes the powerful Embedded Trace Macrocell (ETM) on Cortex M3 and M4 to focus the generated trace stored in the ETB. Trace can be focused on specific areas of code, or triggered by complex events, for example. On the Cortex M0+, the MTB provides simple Instruction Trace using shared SRAM.

This documentation is divided into four parts.

1.1.1 Supported Targets

Instruction Trace is not supported on all targets. It only works on targets that have the necessary hardware. Instruction Trace is currently supported on the following targets from NXP.

- MTB is supported on a range of Cortex-M0+ MCU parts from NXP including:
  - LPC8xx
  - LPC11U6x / LPC11UEx
  - Many Kinetis KL series parts, including MKL43Z

- ETB is supported on a range of Cortex-M3, M4 and M7 MCU parts from NXP including:
  - LPC18xx / LPC43xx

- Many Kinetis K series parts, including MK64F

Note that Instruction Trace is not supported with LPC17xx MCUs (which do not implement an ETB) or LPC13xx/LPC15xx MCUs (which do not implement ETM or ETB).

2. Getting Started

The following tutorials guide you through the process of using Instruction Trace.

2.1 Configuring the Cortex-M0+ for Instruction Trace

Instruction Trace on Cortex-M0+ targets requires some SRAM to be reserved. The Micro Trace Buffer (MTB) that provides the Instruction Trace capabilities stores the execution trace to SRAM. It is therefore necessary to reserve some SRAM to ensure that user data is not overwritten by trace data.

Note:

This configuration is **not** required for Instruction Trace on **Cortex-M3** and **Cortex-M4** MCU parts.

MCUXpresso IDE's new project wizard will automatically include a file called `mtb.c` for parts that support MTB trace. This file places an array called `__mtb_buffer__` into memory at the required alignment. The MTB will then be configured to use this space as its buffer, allowing it to record execution trace without overwriting user code or data. The array `__mtb_buffer__` should not be used within your code, since the MTB will overwrite any data entered into it.

The enablement and size of the `__mtb_buffer__` is controlled by three symbols:

- `__MTB_DISABLE` - If this symbol is defined, then the buffer array for the MTB will not be created.
- `__MTB_BUFFER_SIZE` - a symbol specifying the size of the buffer array for the MTB. This must be a power of 2 in size, and fit into the available RAM. The MTB buffer will also be aligned to its 'size' boundary and be placed at the start of a RAM bank (which should ensure minimal or zero padding due to alignment).
- `__MTB_RAM_BANK` - allows MTB Buffer to be placed into a specific RAM bank. When this is not defined, the "default" (first if there are several) RAM bank is used.

To change or add these symbols, click on Quickstart Panel->Quick Settings->Defined Symbols.

If you have a project for an MTB supported part that does not already contain the `mtb.c` file, you simply need to copy the `mtb.c` file from an existing project, or create a new project and copy the file from there.

Warning

Enabling the MTB manually without properly configuring the target's memory usage will result in unpredictable behavior. The MTB can overwrite user code or data, which is likely to result in a hard fault.

2.2 Trace the Most Recently Executed Instructions

This tutorial will get you started using the Instruction Trace capabilities of MCUXpresso IDE. You will configure the target's trace buffer as a circular buffer, let your program run on your target, suspend it, and download the list of executed instructions.

Note:

Instruction Trace depends on optional components in the target. These components may or may not be available on the LPC device you are working with. See the Instruction Trace Overview [1] for more information.

First, you will debug code on your target. You can import an example project or use one of your own.

Step 0: Configure memory, if using a Cortex-M0+ part If you are using a Cortex-M0+ part with an MTB you must first configure the memory usage of your code to avoid conflicts. See Configuring the Cortex-M0+ for Instruction Trace [3] to learn how to do this.

Step 1: Start the target and show the Instruction Trace View

1. Build and execute your code on the target.
2. Suspend the execution of your target by selecting **Run -> Suspend**.
3. Display the **Instruction Trace** View by clicking 🗗 Window -> Show View -> Instruction Trace.

Step 2: Configure the trace buffer

1. Press the **Record Trace Continuously** button 🔴 in **Instruction Trace**.
2. Resume execution of your target by selecting **Run -> Resume**.

The trace buffer should now be configured as a circular buffer, and your target should be running your code. Once the trace buffer is filled up, older trace data is overwritten by the newer trace data. Configuring the trace buffer as a circular buffer ensures that the most recently executed code is always stored in the buffer.

If the **Record Trace Continuously** button 🔴 was grayed out, or you encountered an error, check out the troubleshooting guide [23].

Step 3: Download the content of the buffer

1. Suspend the execution of your target by selecting **Run -> Suspend**.
2. Press **Download trace** 🔵 in the **Instruction Trace** View.

There may be a short delay as the trace is downloaded from the target and decompressed. Once the trace has been decompressed, it is displayed as a list of executed instructions in the **Instruction Trace** View.

Step 4: Review the captured trace

In this step we explore the captured trace by stepping through it in the instruction View and linking the currently selected instruction to the source code that generated it, as well as seeing it in context in the disassembly View.

1. Toggle the **Link to source** button 🔵 so that it is selected.
2. Toggle the **Link to disassembly** 🔵 so that it is also selected.
3. Select a row in the **Instruction Trace** View.
4. Use the up and down cursor keys to scroll through the rows in the **Instruction Trace** View.

As a row becomes selected, the source code corresponding to the instruction in that row should be highlighted in the source code editor. The disassembly View should also update with the current instruction selected. There can be a slight lag in the disassembly View as the instructions are downloaded from the target and disassembled.

Step 5: Highlight the captured instructions

In this step we turn on the profile View. In the source code editor the instructions that were traced will be highlighted. This highlighting can be useful for seeing code coverage. In the disassembly View each instruction is labeled with the number of times each it was executed.

• Toggle the **Profile information** button 🔵

2.3   Stop Trace When a Variable is Set (Cortex M3 or M4 using ETB)

In this next tutorial you will configure Instruction Trace to stop when the value of a variable is set to a specific value.

This tracing could be useful for figuring out why a variable is being set to a garbage value. Suppose we have a variable that is mysteriously being set to 0xff when we expect it to always be between 0x0 and 0xA for example.

Note – For Cortex-M0+ based systems, this functionality requires additional hardware to be implemented alongside the MTB. Some parts such as the LPC8xx and LPC11U6x families do not provide this.

To trace the instructions that resulted in the write of the unexpected value we are going to have trace continuously enabled with the ETB acting as a circular buffer. Next we will set up a trigger [11] to stop trace being written to the ETB after the value 0xff gets written to the variable we are interested in. The trigger event requires two DWT comparators. One of the comparators watches for any write to the address of the variable and the other watches for the value 0xff being written to any address. The position of the trigger in the trace is set to capture a small amount of data after the trigger and then to stop putting data into the ETB.

Step 1: Start the target and show the Instruction Trace Config View

1. Build and execute your code on the target.
2. Suspend the execution of your target by selecting **Run -> Suspend**
3. Display the **Instruction Trace Config** View by clicking 🖼 Window -> Show View -> Instruction Trace Config
4. Press the refresh button 🔄 in the **Instruction Trace Config** View

Step 2: Find the address of the variable

1. Display the **Disassembly** View by clicking 🖼 Window -> Show View -> Disassembly
2. Enter the variable name into the location search box and hit enter.
3. Copy the address of the variable to clipboard.

Note:
We are assuming that the variable is a global variable.

Step 3: Enable trace

In this step we configure trace to be generated unconditionally in the **Instruction Trace Config** View.

1. In the **Enable trace** section:
   a. Select the **Simple** tab
   b. Select **enable**

Step 4: Enable stalling

To make sure that no packets get lost if the ETM becomes overwhelmed we enable stalling. Setting the **stall level** to 14 bytes mean that the processor will stall when there are only 14 bytes left in the formatting buffer. This setting ensures that we do not miss any data, however, it comes at the cost of pausing the CPU when the ETM cannot keep up with it. See stalling [9] in the Concepts section for more information.

1. Check the **Stall processor** check box

2. Drag the slider to set the **Stall level** to 14 bytes

Step 5: Configure watchpoint comparator

In the first watchpoint comparator choose **data write** in the comparator drop-down box and then enter the address of the variable that you obtained earlier. This event resource will be true whenever there is a write to that address, regardless of the value written.

Step 6: Configure the value written

Select **Data Value Write** from the drop-down, note that data comparators are not implemented in all watch point comparators. Enter the value we want to match, `0xff`, in the text box. Next we link the data comparator to the comparator we configured in **step 5** by selecting 1 in both the **link 0** and **link 1** fields. Select the **Data size** to **word**.

The event resource **Comparator 2** will now be true when **Comparator 1** is true and the word `0x000000ff` is written.

Step 7: Configure the Trigger condition

In the **Trigger condition** section select the **One Input** tab. Set the resource to be **Watchpoint Comparator 2** and ensure that the **Invert resource** option is **not** checked. These setting ensure that a trigger is asserted when the **Watchpoint Comparator 2** is true — i.e. when `0xff` is written to our focal variable.

Step 8: Prevent trace from being recorded after the trigger

Slide the **Trigger position** slider over to the right, so that only 56 words are written to the buffer after the trigger fires. This setting will provide some context for the trigger and allows up to 4040 words of trace to be stored from before the trigger, which will help us see how the target ended up writing `0xff` to our variable.

The configured View should look like Figure 4.3.

Step 9: Configure and resume the target

Now press the green check button ✅ to apply the configuration to the ETM and ETB. Resume the target after the configuration has been applied. Once the target resumes, the buffer will start filling with instructions. Once the buffer is filled the newest instructions will overwrite the oldest. When the value `0xff` is written to the focal variable, the **trigger counter** will start to count down on every word written to the buffer. Once the **trigger counter** reaches zero, no further trace will be recorded, preserving earlier trace.

Step 10: Pause target and download the buffer

After some time view the captured trace by pausing the target and downloading the content of the buffer:

1. Suspend the execution of your target by selecting **Run -> Suspend**
2. Display the **Instruction Trace** View by clicking 🔲 Window -> Show View -> Instruction Trace
3. Download the content of the ETB by pressing 🔽 in the **Instruction Trace** View.
4. Check that the trigger event occurred and was captured by the trace by clicking on 🔳

3. Concepts

3.1 Micro Trace Buffer (MTB)

The CoreSight Micro Trace Buffer for the M0+ (MTB) provides simple execution trace capabilities to the Cortex-M0+ processor. It is an optional component which may or may not be provided in a particular MCU. See *Supported Targets*

The MTB captures Instruction Trace by detecting non-sequentially executed instructions and recording where the program counter (PC) originated and where it branched to. Given the code image and this information about non-sequential instructions, the Instruction Trace component of MCUXpresso IDE Trace is able to reconstruct the executed code.

The huge number of instructions executed per second on a target generate large volumes of trace data. Since the MTB only has access to a relatively small amount of memory, it gets filled very quickly. To obtain useful trace a developer can configure the MTB to only focus on a small area of code, to act as a circular buffer or download the content of the buffer when it fills up. Additionally, all three of these techniques may be combined.

The following sections contain detailed information about the MTB's operation and use.

- Enabling the MTB [7] - options for controlling the MTB
- MTB memory configuration - how the MTB uses memory
- MTB Watermarking [8] - reacting to the buffer filling up
- MTB Auto-resume - combining multiple trace buffer captures

> ⚠ **Warning**
>
> The MTB does not have its own dedicated memory. The memory map used by the target must be configured so that some RAM is reserved for the MTB. The MTB must then be configured to use that reserved space as described in the Configuring the Cortex-M0+ for Instruction Trace [3] in the Getting Started section.

3.1.1 Enabling the MTB

The Micro Trace Buffer (MTB) can be enabled by pressing the **Continuous Recording** button 🔴 or checking **enable MTB** in the **Instruction Trace Config** View. Trace will only be recorded by the MTB when it is enabled.

The MTB can also be enabled and disabled by two external signals TSTART and TSTOP. On the LPC8xx parts, these are driven by the "External trace buffer command" register:

```
// Disable trace in "External trace buffer command" register
LPC_SYSCON->EXTTRACECMD = 2;
 :
 :
// Renable trace in "External trace buffer command" register
LPC_SYSCON->EXTTRACECMD = 1;
```

Note that if TSTART and TSTOP are asserted at the same time TSTART takes priority.

3.1.2 MTB Memory Configuration

Both the size and the position in memory of the MTB's buffer are user configurable. This flexibility allows the developer to balance the trade off between the amount of memory required by their

code and the length of Instruction Trace that the MTB is able to capture before getting overwritten or needing to be drained.

Since the MTB uses the same SRAM used by global variables, the heap and stack, care must be taken to ensure that the target is configured so that the MTB's memory and the memory used by your code do not overlap. For more information see Configuring the Cortex-M0+ for Instruction Trace [3] in the Getting Started Guide.

3.1.3 MTB Watermarking

The MTB watermarking functionality allows the MTB to respond to the buffer filling to a given level by stopping further trace generation or halting the execution of the target. The watermark level and the actions to perform can be set in the **Instruction Trace Config** View . The defined action is performed when the Watermark level matches the MTB's write pointer value. The halt action can be augmented with the **auto-resume** behavior.

3.1.4 MTB Auto-Resume

MCUXpresso Trace provides the option to automatically download the content of the buffer and resume execution when the target is paused by the watermarking mechanism. This **auto-resume** functionality allows extended trace runs to be performed without being constrained by the size of the on chip buffer.

There is a significant performance cost associated with using **auto-resume** since the time taken to pause the target, download the buffer content and resume it again is much greater than the time the MTB takes to fill the buffer.

The data is not decompressed during the auto-resume cycles and so it is still necessary to press **Download trace buffer** in the **Instruction Trace** toolbar to view the captured trace.

Tip:
To suspend the target once **auto-resume** is set press the **Cancel** button in the **downloading trace** progress dialog box. If the **downloading trace** progress dialog box is not displayed long enough to click **Cancel** use the **Stop auto-resume** button in the **Instruction Trace View** toolbar. This disables **auto-resume** so that the target will remain halted the next time the MTB suspends it.

3.1.5 MTB Downloading Trace

To obtain the instruction trace from the MTB once it has captured trace into its buffer, the content of the buffer needs to be downloaded and decompressed. There must be an active debug session for the trace to be downloaded. It can be downloaded using the Download Trace Buffer button in the Instruction Trace View.

The trace recorded by the MTB is compressed. The code image is required to decompress the trace. This image must be identical to the image running on the target when the trace was captured. It is possible to download and decompress a trace from a previous session if the same code image is running on the target. However, if the trace buffer contains old trace data, and a different code image is downloaded to the target, downloading the buffer may result in invalid trace being displayed.

3.2 Embedded Trace Macrocell (ETM)

The Embedded Trace Macrocell (ETM) provides real-time tracing of instructions and data. By combining the ETM with an ETB, some features of this powerful debugging tool are accessible using a low cost debug probe.

This section describes some of the key components and features of the ETM.

3.2.1 Stalling

The ETM uses as relatively small FIFO buffer to store formatted packets, which are then copied to the ETB. This FIFO buffer can overflow when the rate of packets being written into it exceed the rate at which they can be copied out to the ETB.

If stalling is supported on the target the ETM can be set to stall the processor when an FIFO overflow is imminent. The stall level sets the threshold number of free bytes in the FIFO at which the processor should be stalled. The stall does not take effect instantly and so the level should be set such that there is space for further packets to be added to the buffer after the stall level is reached. Overflows can still occur even with stalling enabled.

The maximum value is the total number of bytes in the FIFO buffer. Setting the stall level to this will stall the processor whenever anything is entered into the FIFO. For example if a stall level of 14 bytes was chosen, the processor would be instructed to stall any time there were fewer than 14 bytes left in the in the buffer. If the target had a 24 byte FIFO buffer, this level would allow a 10 byte safety buffer for packets generated between the stall level being detected and the processor actually stalling.

Stalling is enabled from the **Instruction Trace Config** View . Check the **Stall processor** box and select a stall level with the slider.

3.2.2 ETM Events

Events are boolean combinations of two event resources [9]. These events can be used to control when tracing is enabled or to decide when the trigger will occur for example. The available event resources are dependent on the chip vendor's implementation.

Events can be specified in the **Instruction Trace Config** View . For details on how to build these events see ETM event configuration .

3.2.3 Event Resources

Event resources are used by the ETM to control the ETM's operation. They can be combined to form ETM events.

Different sets of event resources are implemented by different silicon vendors. The following event resources are supported by MCUXpresso IDE Trace:

- Watchpoint comparators — match on addresses and data values
- Counters — match when they reach zero
- Trace start/stop unit — combine multiple input
- External — match on external signals
- Hard wired — always match

3.2.4 Watchpoint Comparator Event Resource

Watchpoint comparators are event resources that facilitate the matching of addresses for the PC and the data access as well as matching data values. They are implemented by the Data Watchpoint and Trace [12] (DWT) component.

3.2.5 Counter Event Resource

A target may support a single **reduced counter** on counter 1. The reduced counter decrements on every cycle. This event resource is true when the counter equals zero, it then reloads and

continues counting down. The reload value can be set in the ETM **Instruction Trace Config** View.

3.2.6 Trace Start/Stop Unit Event Resource

The trace start / stop block is an event resource that can combine all of the Watchpoint comparators. For example you could have trace start when the target enters a function call and stop when it exits that function, or use it to generate complex trigger events.

When a start comparator becomes true, the start / stop block asserts its output to high and stays high until a stop comparator becomes true. The start / stop block logic is reset to low when the ETM is reset.

Note that checking both start and stop for the same comparator will be treated as just checking start.

To use this method

* Set the trace enable to use the start stop block
* Find the entry and exit addresses for the focal function (in the disassembly View)
* Create instruction match conditions in the DWT comparators for those addresses
* Check the start box for the entry address and stop boxes for the exit address

3.2.7 External Event Resource

External input may be connected to the ETM by the silicon vendor. Please see their documentation for more information about these vendor configurable elements.

3.2.8 Hard Wired Event Resource

This resource will always be observed to be true. It can be useful for enabling something constantly when used with an **A** function, or to disable something when used with a **NOT A** function.

3.3 Embedded Trace Buffer (ETB)

The Embedded Trace Buffer (ETB) makes it possible to capture the data that is being generated at high speed in real-time and to download that data at lower speeds without the need for an expensive debug probe. MCUXpresso IDE Instruction Trace on Cortex-M3 and Cortex-M4 targets is facilitated by the ETB in conjunction with the ETM.

The ETB and ETM are optional components in the Cortex-M3 and Cortex-M4 targets. Their implementation is vendor specific. When they are implemented the vendors may implement different subsets of the ETM's and ETB's features. Instruction Trace will automatically detect which features are implemented for your target; however, note that not all of the features listed in this guide may be available on your target.

The ETM compresses the sequence of executed instruction into packets. The ETB is an on chip buffer that stores these packets. This tool downloads the stored packets from the ETB and decompresses them back into a stream of instructions.

This feature is useful for finding out how your target reached a specific state. It allows you to visualize the flow of instructions stored in the buffer for example.

There are multiple ways to use the ETB. The simplest is continuous recording, where the ETB is treated as a circular buffer, overwriting the oldest information when it is full. There are also more advanced options that allow the trace to be focused on code that is of interest to make the

most of the ETB's memory. This focusing is achieved by stopping tracing after some trigger or by excluding regions of code. These modes of operation are described in this document.

3.3.1 Triggers

The trigger mechanism of using Instruction Trace works by constantly recording trace to the ETB until some fixed period after a specified event. This method allows the program flow up to, around or after some event to be investigated.

There are two components that need to be configured to use triggers. These are the trigger condition and the trigger counter. The trigger condition is an event (a Boolean combination of event resources). When the trigger condition event becomes true, the trigger counter counts down every time a word is written to the ETB. When the trigger counter reaches zero no further packets are written to the ETB.

To use the ETB in this way the Trace Enable event is set to be always on (hard wired). The ETB is constantly being filled, overflowing and looping round, overwriting old data. The trigger counter is set using the trigger position slider. The counter's value is set to the **words after trigger** value. You can think of the position of the slider as being the position of the trigger in the resulting trace that will be captured by the ETB.

There are three main ways of using the trigger: trace after – when that the data from the trigger onwards is the interesting information; trace about — the data either side of the trigger is of interest; and trace before — data before the trigger is the key information.

Trace after

When the slider is at the far left, the **words before trigger** will be zero and the **words after trigger** will be equal to the number of words which can be stored in the ETB. This corresponds to the situation where the region of interest occurs after the trigger condition. Once the ETB receives the trigger packet the trigger counter, which was equal to **words after trigger** will count down with every subsequent word written to the ETB, until it reaches zero. The trigger packet will be early in the trace and the instruction trace will include all instructions from when the trigger event occurred, until the ETB buffer filled up.

Note that in order to facilitate decoding of the trace the ETM periodically emits synchronization information. On some systems the frequency that these are generated can be set. Otherwise, by default, they are generated every `0x400` cycles. It is therefore necessary to make sure that you allow some trace data to be collected before your specific area of interest when using the trigger mechanism so that this synchronization information is included.

Trace around

As the trigger position slider is moved to the right, the **words after trigger** value decreases. This means that the data will stop getting written to the ETB sooner. Since the Trace Enable event was set to true, there will be older packets, from before the trigger event, still stored in the ETB. The resulting instruction trace will include some instruction from before the trigger and some from after the trigger. Use the slider to balance the amount of trace before and after the trigger.

Trace before

With the trigger position slider towards the far right, the instruction trace will focus on the trace before the trigger event. Note that only trace captured after the instruction trace has been configured will be captured. For example pausing the target and setting a trigger condition for the next instruction, with the trace position set to the far right would not be able to include instruction trace from before the trace was configured, but rather it would stop after **words after trigger** number of words was written to the ETB, leaving most of the ETB unused.

Note that setting the trigger position all the way to the right, such that **words after trigger** is zero will disable the trigger mechanism.

3.3.2 Timestamps

When the **timestamps** check-box is selected and implemented, a time stamp packet will be put into the packet stream. The interval between packets may be configurable if the target allows it, or may be generated at a fixed rate.

If supported, there is a Timestamp event. Timestamps are generated when the event fires. A counter resource could be used to periodically enter timestamps into the trace stream for example.

Note:

ARM recommends against using the **always true** condition as it is likely to insert a large number of packets into the trace stream and make the FIFO buffer overflow.

Note that a time of zero in the time stamp indicates that time stamping is not fully supported

3.3.3 Debug Request

The ETB can initiate a debug request when a trigger condition is met. This setting causes the target to be suspended when a trigger packet is created.

Note

There may be a lag of several instructions between the trigger condition being met and the target being suspended

3.3.4 Output All Branches

One of the techniques used by the ETM to compress the trace is to output information only about indirect branches. Indirect branches occur when the PC (program counter) jumps to an address that cannot be inferred directly from the source code.

The ETM provides the option to output packets for all branch instructions — both indirect and direct. Checking this option will output a branch packet for every branch encountered. These branch packets enable the reconstruction of the program flow without requiring access to the memory image of the code being executed.

This option is not usually required as the Instruction Trace tool is able to reconstruct the program flow using just the indirect branches and the memory image of the executed code. This option dramatically increases the number of packets that are output and can result in FIFO overflows, resulting in data loss or reduced performance if stalling [9] is enabled. It can also make synchronization harder (e.g. in triggered traces) as you can end up with fewer I-Sync packets in the ETB.

3.4 Data Watchpoint and Trace

The data watchpoint and trace (DWT) unit is an optional debug component. Instruction Trace uses its watchpoint to control trace generation. The MCUXpresso IDE **Data Watch** View uses it to monitor memory locations in real-time, without stopping the processor.

Warning:

Instruction Trace and SWO Data Watch Trace cannot be used simultaneously as they both require use of the DWT unit.

3.4.1 Instruction Address Comparator

Use the program counter (PC) value to set a watchpoint comparator resource true when the PC matches a certain instruction. Choose **Instruction** from the comparator drop-down and enter

the PC to match in the match value field. Use the Disassembly View to find the address of the instruction that you are interested in. When the PC is equal to the entered match value, the watchpoint comparator will be true; otherwise it is false.

Setting a mask enables a range of addresses to be matched by a single comparator. Set the **Mask size** to be the number of low order bytes to be masked. The range generated by the mask is displayed next to the **Mask size** box. The comparator event resource will be true whenever the PC is within the range defined by the mask.

Note:
Instruction addresses must be half-word aligned

Warning
Instruction address comparators should not be applied to match a `NOP` or an `IT` instruction, as the result is unpredictable.

Data Access Address Comparators

Data access address Comparators are event resources that watch for reads or writes to specific addresses in memory. As with the instruction comparator, the address is entered as the match value. There are three different data access comparators:

- Data R/W — true when a value is read from or written to the matched address
- Data Read — only true if a value is read from the matched address.
- Data Write — only true if a value is written to the matched address.

Like instruction address comparators, data access address comparators can operate on a range of addresses.

Note:
These comparators do not consider the value being written or read — they only consider the address that is being read from or written to.

3.4.2 Data Value Comparator

Data value comparators are triggered when a specified value is written or read, regardless of the address of the access. This comparator is typically implemented on one of the Watchpoint comparators on Cortex chips. There are three types of this comparator:

- Data Value R/W — true when a value is read or written that is equal to the **Match Value**
- Data Value Read — only true if a value is read that is equal to the **Match Value**
- Data Value Write — only true if a value is written that is equal to the Match Value

The size of the value to be match must be configured as either a **word**, **half word** or **byte** in the **Data size** drop-down. Only the lowest order bits up to the request size will be matched. For example, if the **Data size** is set to **byte**, only the lowest order byte of the match value will be used in the comparisons.

Typically, you might want to match only a specific value written to a specific variable. Data value comparators provide this facility by linking to up to two data access address comparators:

- Access to any address
  - set **link 0** and **link 1** to the data value comparator number
- Access to one address specified in another DWT comparator
  - Set both **link 0** and **link 1** to the address match comparator number
- Access to either of two addresses specified in two separate DWT comparators

- Set **link0** to the first address comparator id
- Set **link1** to the second address comparator id

Tip:
When there are only two DWT comparators the option to link the two comparators is given as a check-box.

3.4.3 Cycle Count

If supported by the chip vendor, the first comparator can implement comparison to the **Cycle Counter**. The **Cycle Counter** is a 32-bit counter which increments on every cycle and overflows silently. This event resource is true when the cycle counter is equal to the match value.

To use this feature, choose **Cycle Counter** from the comparator drop-down and enter the match value into the **Match Value** field.

4. Reference

4.1 Instruction Trace View
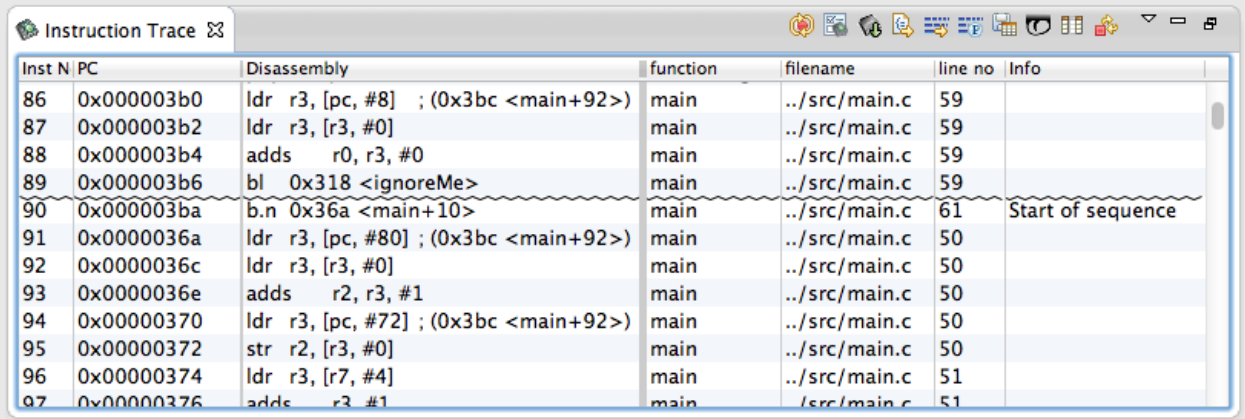
From the **Instruction Trace View** you can configure trace for your target, and download and view the captured trace. Open the **Instruction Trace View** by clicking 🔁 Window -> Show View -> Instruction Trace.

It should look like Figure 4.1.

For trace generated by the ETM, the color of the text in the instruction list provides information about the traced instruction. Grayed-out text indicates that the instruction did not pass it condition. A red background indicates a break in the trace due an ETM FIFO buffer overflow. Instructions may be missing between red highlighted instruction and the proceeding entry in the trace View. If the **Stall** option is available it can be used to help ensure this does not occur in subsequent traces.

A break in the trace may occur due to trace becoming disabled and then renabled (for example to exclude the tracing of a delay function). Breaks in the trace are indicated by a line drawn across the row.

A green background highlights the trigger packet that is generated after the trigger condition is met. Press the **trigger button** to jump to this instruction in the instruction list.

Inst N	PC	Disassembly	function	filename	line no	Info
86	0x000003b0	ldr r3, [pc, #8] ; (0x3bc <main+92>)	main	../src/main.c	59	
87	0x000003b2	ldr r3, [r3, #0]	main	../src/main.c	59	
88	0x000003b4	adds r0, r3, #0	main	../src/main.c	59	
89	0x000003b6	bl 0x318 <ignoreMe>	main	../src/main.c	59	
90	0x000003ba	b.n 0x36a <main+10>	main	../src/main.c	61	Start of sequence
91	0x0000036a	ldr r3, [pc, #80] ; (0x3bc <main+92>)	main	../src/main.c	50	
92	0x0000036c	ldr r3, [r3, #0]	main	../src/main.c	50	
93	0x0000036e	adds r2, r3, #1	main	../src/main.c	50	
94	0x00000370	ldr r3, [pc, #72] ; (0x3bc <main+92>)	main	../src/main.c	50	
95	0x00000372	str r2, [r3, #0]	main	../src/main.c	50	
96	0x00000374	ldr r3, [r7, #4]	main	../src/main.c	51	
97	0x00000376	adds r3, #1	main	../src/main.c	51	

Figure 4.1. The Instruction Trace View

4.1.1 Instruction Trace View Toolbar Buttons

There are several buttons in the toolbar of the **Instruction Trace View** that allow you to use Instruction Trace with your target.

- 🔴 Record trace continuously
- 🖼 Show Instruction Trace configuration View
- 📥 Download trace buffer from target
- 📄 Link to Source
- 📊 Link to Disassembly View
- 📊 Show profile information

- Save trace to csv

- Jump to trigger

- Stop auto resume

- Select columns to display

4.1.2 Record Trace Continuously

The Record trace continuously button configures the trace buffer as a circular buffer. Once the trace buffer is filled up, older trace data is overwritten by newer trace data.

This mode of operation ensures that when the target is paused, the buffer will contain the most recently executed instructions.

Note: The target must be connected, with your code downloaded and execution suspended, before you can configure trace.

4.1.3 Show Instruction Trace Config View

Press the Show Instruction Trace config View button to display the **Instruction Trace config** View. This View will provide you with access to all of the trace buffer's configuration settings.

The **Instruction Trace config** View's contents depend on the features supported by your target. See the following sections for more information on your target.

- Cortex M0+ MTB
- Cortex M3 ETB
- Cortex M4 ETB

Note: The target must be connected, with your code downloaded and execution suspended, before you can configure trace.

4.1.4 Download Trace Buffer

Press the Download trace buffer button to download the content of the trace buffer from the target. The data will be downloaded and decompressed. The list of executed instructions will be entered into the Instruction View table.

Notes

- The target must be suspended in order to perform this action
- The content of the trace buffer can persist across resets on some targets. The buffer is decompressed using the current code image. If the code has changed since the data was entered in the buffer, the decompresser's output will be garbage.
- If no instructions are listed after downloading, check your configuration to make sure that Instruction Trace started.

4.1.5 Link to Source

The Link to source toggle button enables the linking of the currently-selected instruction in the Instruction Trace View to the corresponding line of source code in the source code viewer.

The line of source code that generated the selected instruction will be highlighted in the source code viewer.

Tip:
You can use the the cursor keys within the Instruction Trace View to scroll through the executed instructions.

4.1.6 Link to Disassembly

The Link to disassembly toggle button enables the linking of the currently selected instruction in the Instruction Trace View to the corresponding instruction in the disassembly viewer. The selected instruction will be highlighted in the disassembly viewer.

Note: The target must be suspended to allow the disassembly View to display the code on the target.

Tip:
You can use the the cursor keys within the Instruction Trace View to scroll through the executed instructions.

4.1.7 Show Profile Information

The Toggle profile button enables and disables the display of profile information corresponding to the currently downloaded instruction trace.

When the display of profile information is enabled, a column appears in the disassembly View that shows the count of each executed instruction that was captured in the trace buffer.

Lines of source code whose instructions were recorded in the trace buffer are highlighted in the source View.

Tip:
You can customize the display of the highlighting by editing the **Profile info in source View** item in the preferences panel under General -> Editors -> Text Editors -> Annotations

4.1.8 Save Trace

The Save trace button saves the content of the **Instruction Trace** View to a csv file. These files can be large and may take a few seconds to save.

4.1.9 Jump to Trigger

Trace downloaded from an ETB (the embedded trace buffer on the Cortex M3 and M4) may contain a trigger packet. If the trace stream contain such a packet, the **jump to trigger** button will show it in the **Instruction Trace** View.

4.1.10 Stop Auto-Resume

When using the MTB auto-resume feature [8], the user may not have time to press the suspend button or the **Cancel** button in the download trace progress dialog box as the target is being

rapidly suspended and restarted. Pressing the **Stop auto-resume** button 🔴 will turn off the auto-resume feature, so that the target will suspend the next time the MTB reaches its watermark level without resuming.

4.1.11 Select Columns ⊞

You can choose the columns shown in the instruction list using the select columns action ⊞. The available columns are listed below.

Rearrange the column ordering in the table by dragging the header of the columns.

Table 4.1. Instruction View column descriptions

Column	Description
Inst No	The index of the instruction in the trace
Time	The timestamp associated with an instruction
PC	The address of the instruction
Disassembly	The disassembled instruction
C	The condition code for the instruction. E for instructions that passed their condition and were executed, N for instructions which were not executed.
opCode	The op code for the instruction.
Arguments	The arguments for the op code
Offset	Offset of the instruction within the function
Function	The C function name which the instruction belongs to
Filename	The C source file that the instruction is associated with
Line no	The line number in the C source file that the instruction is associated with

4.2 Instruction Trace Config View for the MTB

Instruction Trace with the MTB can be fully configured using the **Instruction Trace Config** View. Open the **Instruction Trace Config View** by clicking 🔷 Window -> Show View -> Instruction Trace or by clicking on the **Instruction Trace Config** button 🔲 in the **Instruction Trace** View. Once the target is connected, refreshing the View with the refresh button 🔄 will display the options for the target.

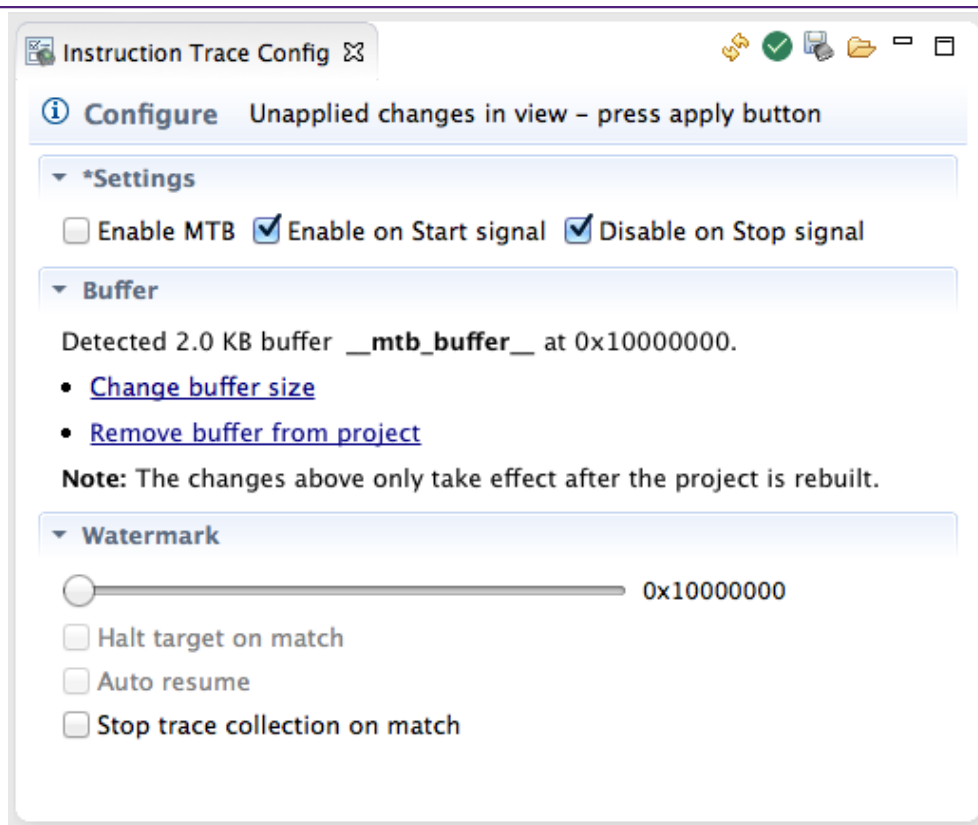Changes made to the MTB configuration are only applied when the **Apply** button ✅ is pressed.

Figure 4.2. Instruction Trace Config View for MTB Instruction Trace

4.2.1 Configuring the Buffer

If the target does not have a buffer allocated for the MTB the View will display instruction on how to create the buffer when it is refreshed.

4.2.2 Enabling

The three check-boxes in the top section of the View control whether the MTB is enabled or not. The first check-box **Enable MTB** can be used to directly enable or disable the MTB. The second two check-boxes control whether or not the MTB is affected by start and stop signals TSTART and TSTOP which can be triggered by software using the target's external trace buffer command register EXTTRACECMD.

4.2.3 Buffer

The buffer section of the MTB configuration View displays information about where in memory the MTB data is stored. It displays the size of the buffer and provides instructions on how to change or remove the buffer.

See *MTB memory configuration* for more information and *Configuring the Cortex-M0+ for Instruction Trace* [3] for an example.

4.2.4 Watermark

The watermark section of the MTB advanced configuration View allows you to configure an action to be performed when the buffer fills to a certain level. The slider configures the watermark level

at which the action is triggered. The address next to the slider indicates the absolute address of the watermark.

Selecting **Halt target on match** suspends the target when the buffer fills to the specified watermark level. When the target is halted the content of the buffer is automatically drained and stored. The buffer is reset so that it can be refilled once the target is resumed. The trace is not decompressed or displayed in the **Instruction Trace** View until the **Download buffer** button is pressed. This behavior allows multiple trace runs to be concatenated.

Selecting **Auto resume** allows the target to be automatically restarted once the buffer has been downloaded. It only has an effect if **Halt target on match** is also selected. This auto-resume feature allow the trace capture not to be limited to the size of the MTB's buffer allowing code coverage to measured. The frequent interruptions have a large impact on target performance.

> ### Note
> You can suspend an auto-resuming target by pressing the **cancel** button in the buffer download progress dialog box. If the MTB buffer is sufficiently small, then the progress dialog box may not be displayed long enough for the user to select **cancel**. In that case you should press the **Stop auto-resume** button . Both of these methods will turn off the auto-resume feature and the target will suspend without restarting the next time the watermark matches.

Selecting **stop trace collection on match** allows the MTB to stop recording trace once it has been filled once, without interrupting the execution of the target. This feature could be useful when used in conjunction with a DWT comparator that starts trace on a certain condition.

4.2.5 Viewing the State of the MTB

The state of the target is read each time the refresh button in the **Instruction Trace Config** View. For example the, the **Enable MTB** box will show whether or not the MTB is currently enabled. This information can be useful for confirming that TSTART and TSTOP signals are affecting the MTB as expected when using the target's external trace buffer command register EXTTRACECMD.

Pressing the **Apply** button will update the MTB's configuration — even if no settings are changed by the user. This action will have the effect of clearing the content of the MTB's buffer. That is, if the MTB contains trace that has not been downloaded and then the user applies the configuration, the content of the buffer will be lost.

4.2.6 Instruction Trace Config View for the ETB

Instruction Trace with the ETB and ETM can be fully configured using the **Instruction Trace Config** View. Open the **Instruction Trace Config View** by clicking Window -> Show View -> Instruction Trace or by clicking on the **Instruction Trace Config** button in the **Instruction Trace** View. Once the target is connected, refreshing the View with the refresh button will display the options for the target.

Changes made to the MTB configuration are only applied when the **Apply** button is pressed. A star appended to a section title indicates that it contains unapplied changes.
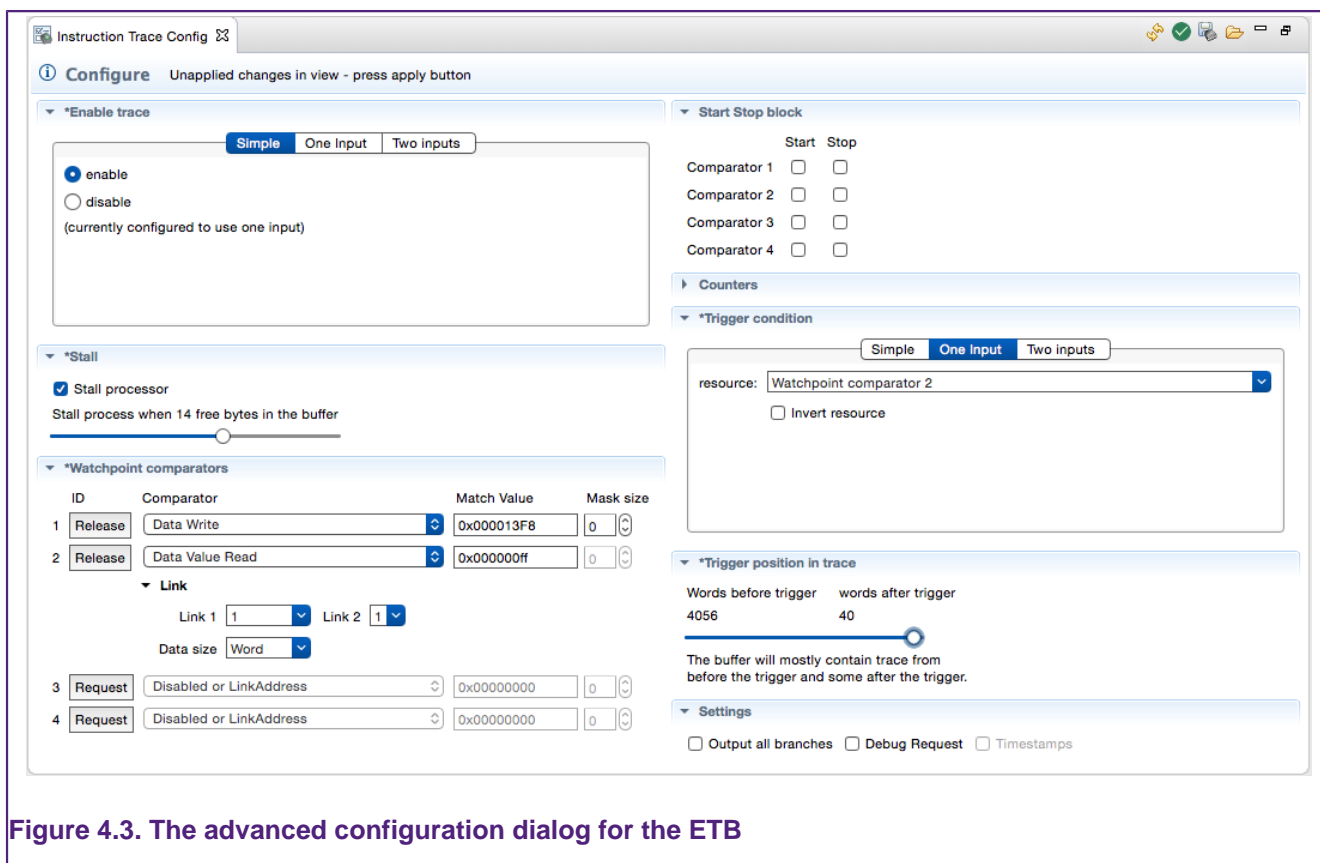
Figure 4.3. The advanced configuration dialog for the ETB

4.2.7 ETM Event Configuration

An ETM event is a boolean combination of up to two event resource inputs [9]. The **Instruction Trace Config** View provides an easy way to build these events by allowing the user to choose the complexity of the event. These are used for trace enablement and the trigger condition for example.

- The simplest option is a binary enabled/disabled choice. This is accessed by selecting the **Simple** tab. Select **enable** for the Event to always be true or **disable** for the event to always be false. See Figure 4.4.

- To use a single event resource select the **One Input** tab. An event resource can be chosen from the drop-down and the event will be true when the event resource is true. Checking the **Invert resource** box will cause the Event to be true when the event resource is false, and visa-versa. See Figure 4.5.

- To combine two event resources select the **Two Inputs** tab. The events can be chosen from the drop-downs and the logical combination operation selected. As with the **One Input** tab the resources can be inverted. See Figure 4.6.

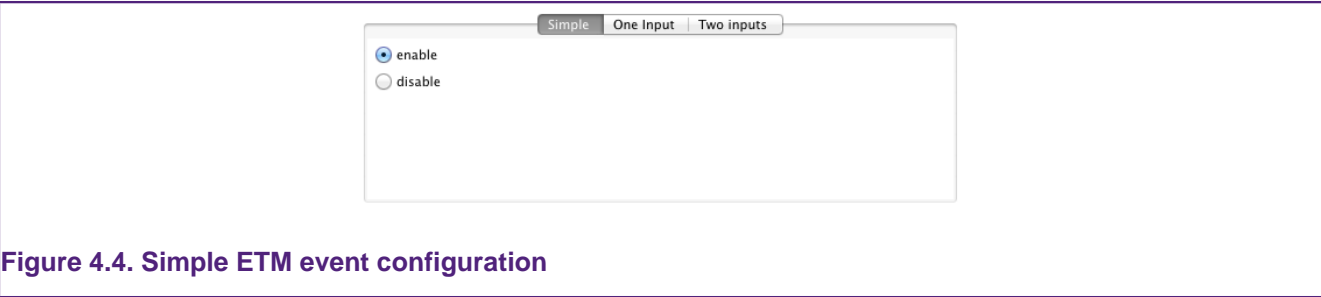The configuration on the visible tab is used when the **Apply** button is clicked.
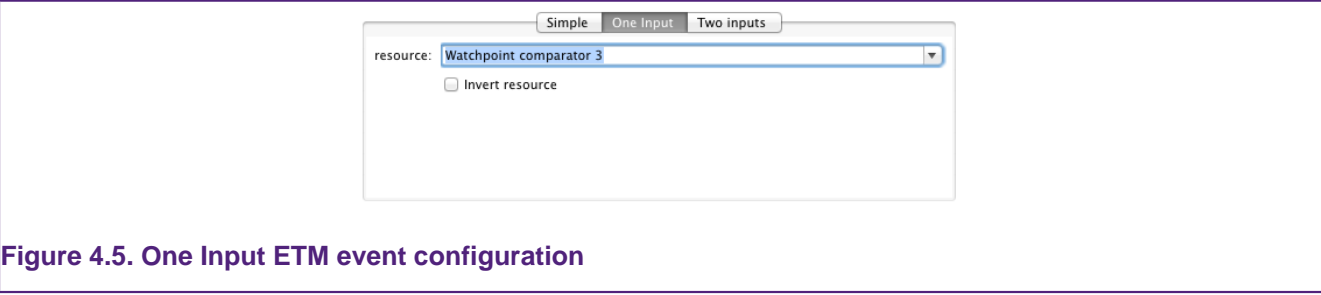
Figure 4.4. Simple ETM event configuration

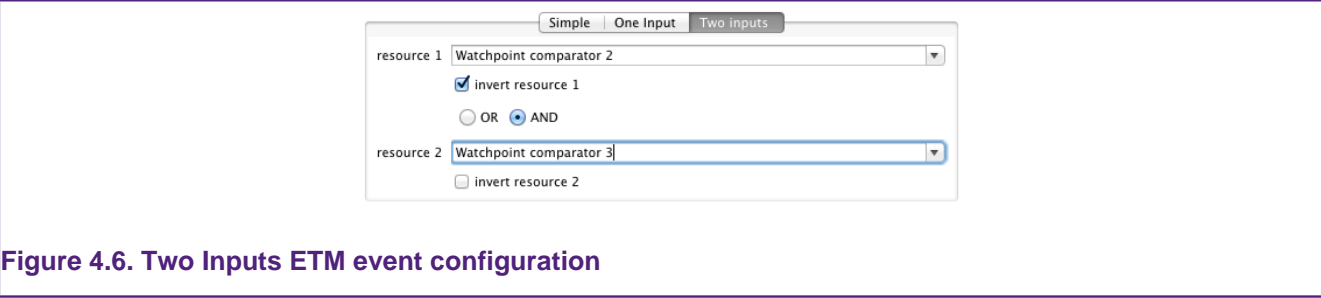**Figure 4.5. One Input ETM event configuration**

Figure 4.6. Two Inputs ETM event configuration

5. Troubleshooting

This section of the help provides solutions to common problems that you may encounter while using Instruction Trace.

5.1 General

5.1.1 Instruction Trace Claims Target Not Supported When it Should Be

Instruction Trace caches some information about the target in the Launch configuration to reduce setup time. In some cases this information may become corrupted. Deleting the Launch configuration will force Instruction Trace to refresh this information.

You can also double check that your target is in *Supported Targets*

5.2 MTB

5.2.1 Target Crashes When MTB is Enabled

The MTB may be overwriting code or data on the target. Check that the MTB's memory configuration is compatible with the target's memory configuration.

5.3 Target Keeps Resuming Itself and I Cannot Stop It

Press the **Stop auto-resume** button in the **Instruction Trace View** toolbar. This button disables **auto-resume** so that the target will remain halted the next time the MTB suspends it.

See *Auto-resume* in the Concepts section for more information.

5.4 ETB

5.4.1 Trigger Packet Missing from Trace Even Though Trigger Occurred

It may be that the trigger packet was lost due to FIFO overflow and was not written to the ETB. To make sure that it actually was triggered, look at the trigger counter (the number of words to write after the trigger condition). The trigger counter only decrements after the trigger has been activated. If it has not decremented, check your trigger condition.

If the trigger counter has decremented and you see no packet, try enabling stalling if it is implemented.

5.5 Comparator Sharing

Watchpoints, Data Watch trace and Instruction Trace all make use of hardware debug comparators. These comparators can match PC values, data values and address accesses which can be used to pause the execution of a target; output a message over the SWO; or to start an Instruction Trace buffer recording for example.

There are a limited number of these hardware debug comparators implemented on a part, for example the DWT unit of a Corex-M3 would typically have four. A specific comparator can only be used by one component at a time. To use a comparator for a particular component,

either watchpoint, data watch or Instruction Trace, it must be requested from the IDE. If all of the comparators are currently in use the request will be denied. Comparators need to be explicitly released by the component that has successfully requested it to be available to another component. Different comparators may be used by different components at the same time.

5.5.1 Watchpoints – Requesting and Releasing Comparators

Watchpoints are similar to breakpoints, but instead of halting execution on a specific line number, they halt execution when a specific variable is accessed.

Comparators for watchpoints are requested automatically when the debugger resumes execution of the target. If there are insufficient comparators available the error message in the figure Figure 5.1 below will be shown.

The comparators for watchpoints are released when the target suspend execution
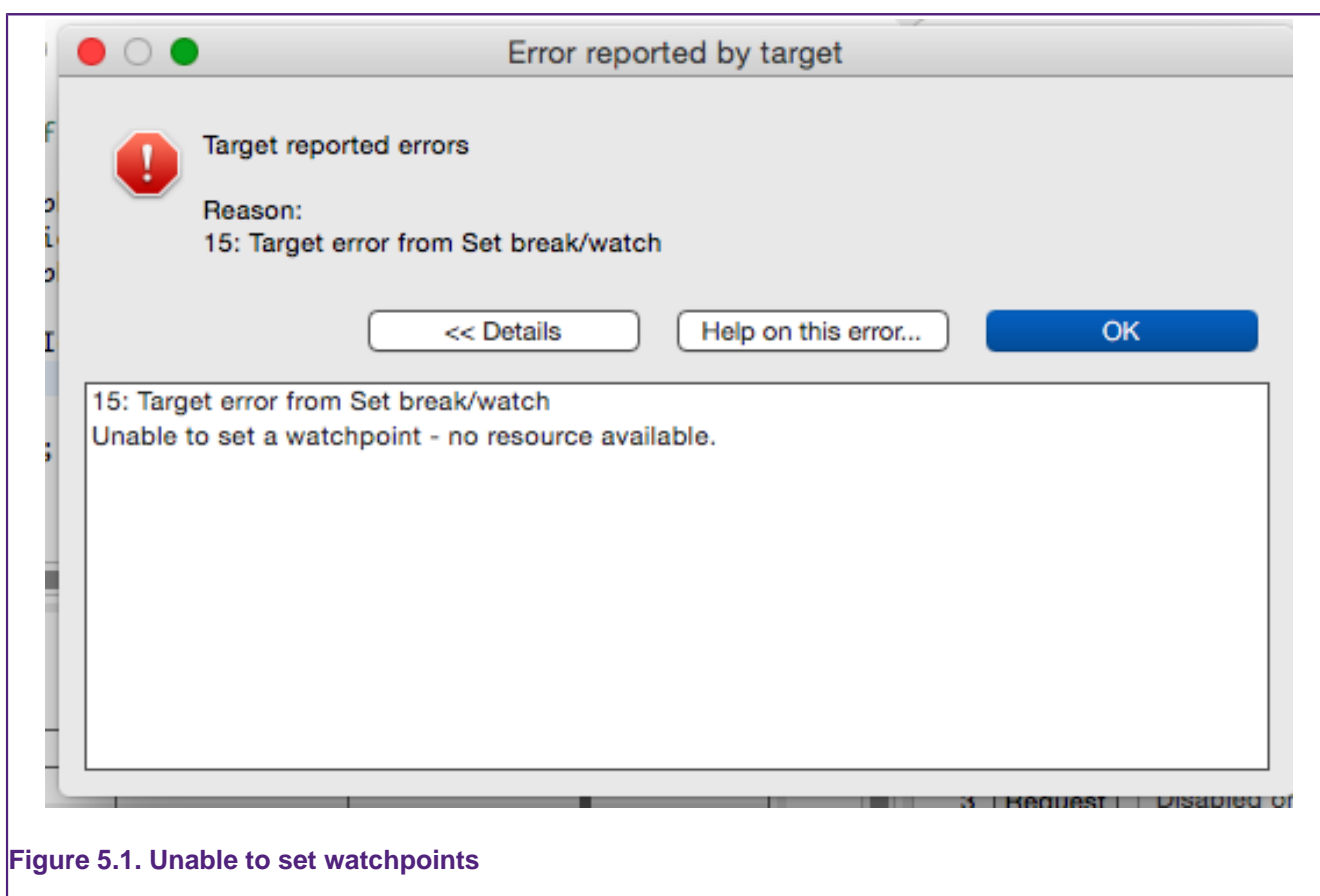

Figure 5.1. Unable to set watchpoints

5.5.2 SWO Data Watch Comparators

The SWO data watch comparators emit information about a data access over the SWO channel.

A comparator is requested when a check is marked next to an item in the SWO Data Watch View. The tool selects the first free comparator to use. If no comparators are available you will be alerted with the error message in figure Figure 5.2.

When an item is unchecked in the SWO Data Watch View its comparator is released and will become available to other components.

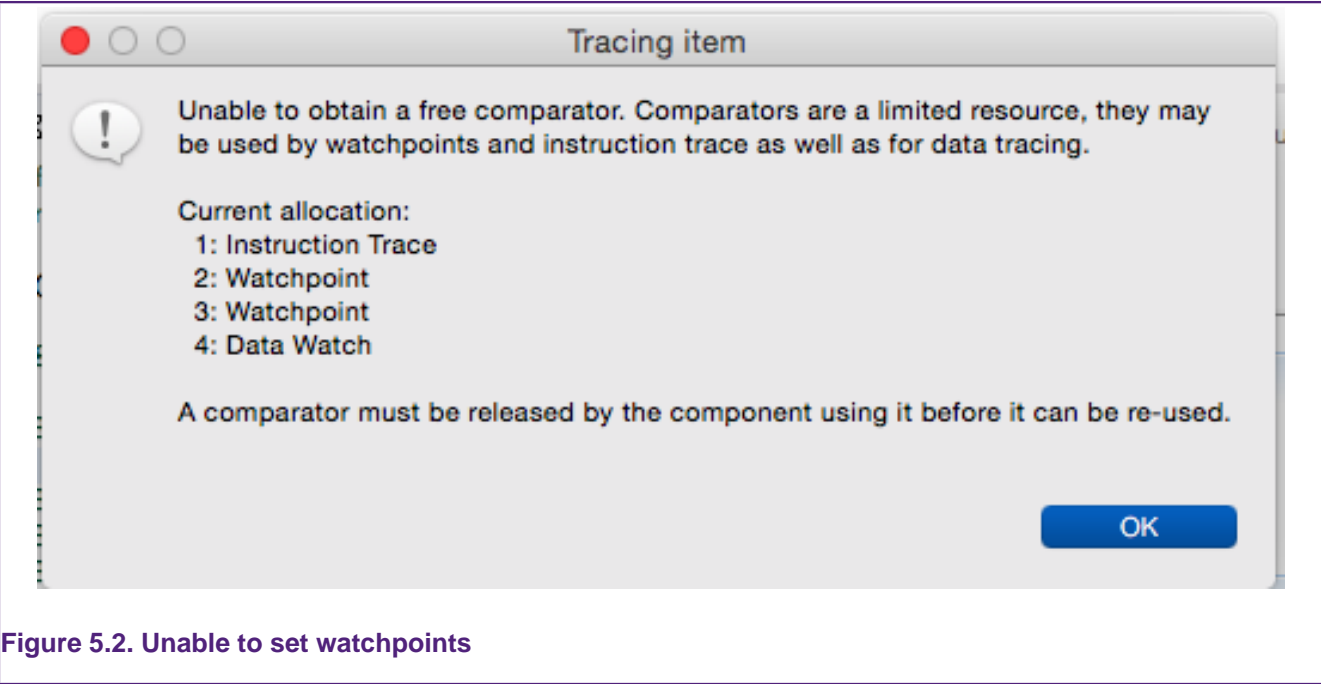See SWO Trace View for more information about using SWO Data Watch Trace.

User Guide   Rev. 10.0.2 — 6 July, 2017   24

Figure 5.2. Unable to set watchpoints

5.5.3 Instruction Trace

The comparators can be used to control the starting and stopping of instruction trace buffers.

To use a comparator select the **Request** button corresponding to the comparator to use in the **Watchpoint comparators** section (see figure Figure 5.3). If that specific comparator is already in use you will see an error message (see figure Figure 5.4). Once it has been successfully requested it can be configured and the **Request** button is replaced with a **Release** button.
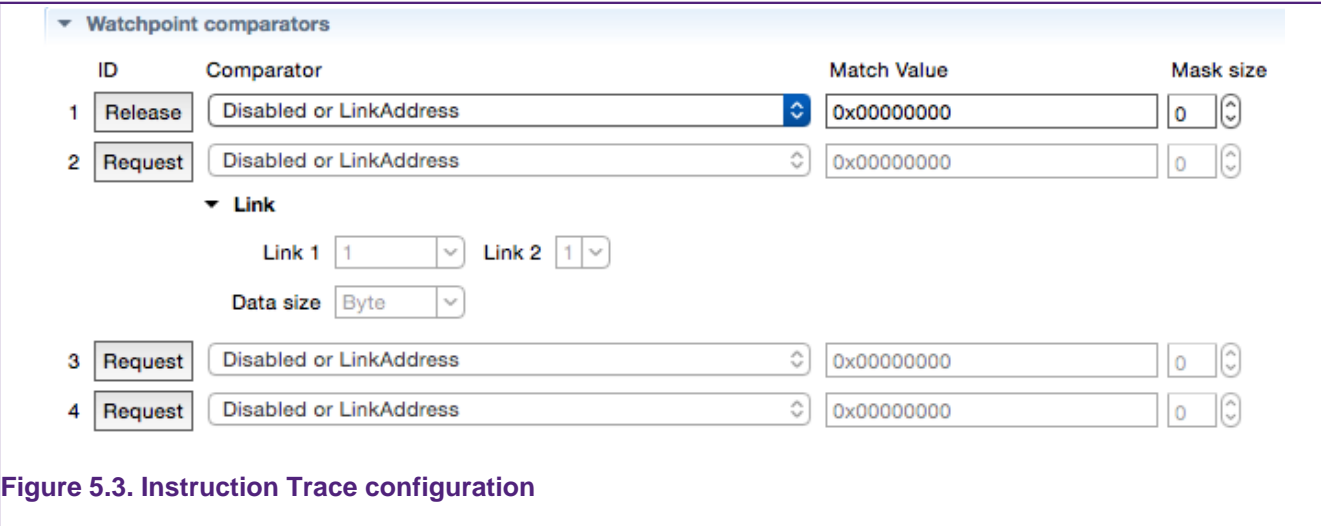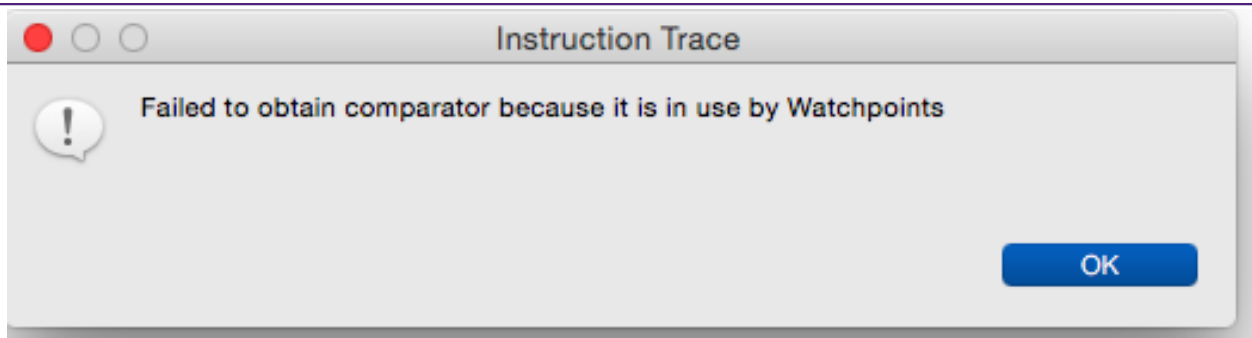

Figure 5.3. Instruction Trace configuration

Press the **Release** button to stop using the comparator for Instruction Trace and allow other components to use it.

Error message shown when the user tries to request a comparators that is already in use.

Figure 5.4. Unable to set watchpoints