AN3290
7/2006

A Guide for Nexus Tracing on Argon

By Michael Fleischer

1 Purpose

The Starcore DSP core used on the Argon Plus and LV SoC's from Freescale Semiconductor provides numerous program and data tracing capabilities through dedicated on-chip hardware modules. This hardware is enabled through the CodeWarrior for Starcore/SDMA 1.0 development and debugging tool's project settings interface. The typical configuration for Nexus Tracing is shown in Figure 1: Nexus Tracing Configuration.

CONTENTS

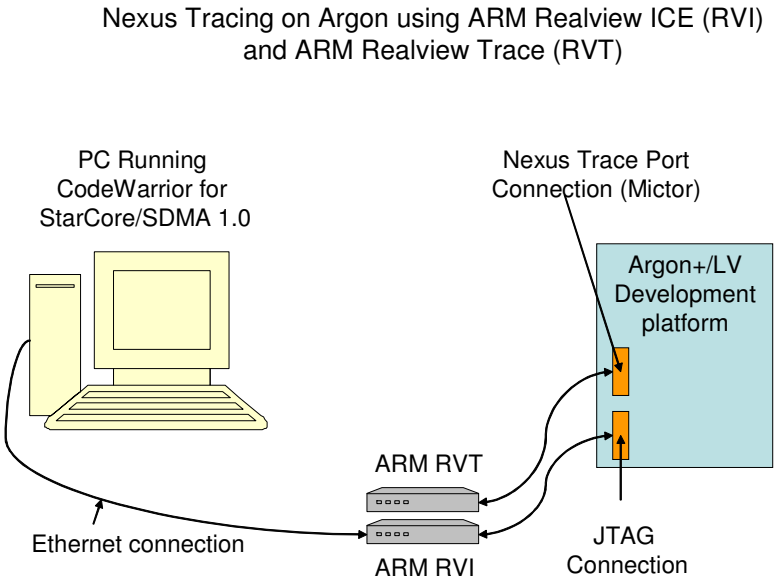

Figure 1: Nexus Tracing Configuration

2 Nexus Hardware Overview

The following chapter describes the Nexus hardware present to support real time tracing for Starcore code execution. The Argon chipset contains a nexus hardware trace module. This module takes multiple inputs for generating trace data either continuously or on a triggered basis. The nexus module takes every input and generates Nexus Messages which are sent out the physical port pins into a trace hardware buffer. These messages are generated when the Nexus module is enabled and also based on the Nexus module's settings.

SC140 Nexus

This is the core module for tracing on the Starcore DSP processor core. The nexus hardware module works by creating and sending message packets into a trace buffer or to an external port for extraction through a Trace port. Control of the Nexus interface can be through embedded software or through JTAG register settings. This document will describe the latter type of control using the CodeWarrior tool.

The Nexus hardware module provides the following capabilities:

- Instruction Tracing: The CodeWarrior tool interpolates execution between program discontinuities for static program tracing. Supported instructions include:

  - Direct branches (BF, BRA, BREAK, BSR, BT, CONT, JF, JMP, JSR, JT, SKIPLS, TRAP)
  - Indirect branches (CONT, JF, JMP, JSR, JT)
  - Hardware Loops (LPMARKA, LPMARKB)
  - Returns (RTE, RTERI, RTS, RTSTK)
- Data Tracing (reads and writes):
  - 32 bit start and end addresses
  - Inclusive of address range and endpoints
  - Exclusive of address range and endpoints
- Data Tracing (Acquisition Channel: Captures writes to a specific address)
- Ownership
  - Process ID comparison
  - Data ID comparison
  - Privilege level (Supervisor or User)
- Watchpoint Trigger 1 for Program tracing can trigger start and end of tracing from:
  - EOnCE module's 6 Program Watchpoints (EDCA0-5)
  - EOnCE module's 1 Data Watchpoint (EDCD)
- Watchpoint Trigger 2 for Program tracing can trigger start and end of tracing from:
  - Four External Watchpoints (routed from ECT): WP1-4
  - Process ID Match or Mismatch
- Watchpoint Trigger 1 for Data tracing can trigger start and end of tracing from
  - EOnCE module's 6 Program Watchpoints (EDCA0-5)
  - EOnCE module's 1 Data Watchpoint (EDCD)
- Watchpoint Trigger 2 for Data tracing can trigger start and end of tracing from
  - Four External Watchpoints (routed from ECT): WP1-4
  - Process ID Match or Mismatch
- Core Profiling
  - Six Profiling counters in two sets of three (DPU-A and DPU-B)

All Nexus packets can be configured with or without a timestamp. Adding a timestamp increases message packet sizes by 24 bytes each.

Nexus Port Controller

The Nexus Port controller is a hardware module used as an arbitrator to configure the Nexus module for use with other modules in the Argon System on Chip. Configuration options include:

- Nexus Output Port mode: Defines Nexus Message port width: All variable length messages are port aligned (data is extended to integer multiple of port width)

  - Reduced Port mode 1: 8 pin Nexus Message Port
  - Reduced Port mode 2: 16 pin Nexus Message Port

- Nexus Clock Control: Divides Nexus Clock from System Clock, must run Nexus interface no faster than slowest Nexus client (AHB or StarCore)
  - System Clock
  - System Clock/2
  - System Clock/4
  - System Clock/8
- MSEO Configuration: Controls Message Start and Message End signaling for variable length message packets
  - 1 bit MSEO control provides a single pin interface
  - 2 bit MSEO control provides slightly improved message signaling and is the recommended setting to be used with an ARM RVT
- Data Rate Enable Control
  - Single Data Rate Control-Nexus messages clocked on rising edge of Nexus clock (Compliant with IEEE-ISTO 5001-2003 standards)
  - Double Data Rate Control-Nexus messages clocked on rising and falling edge of Nexus clock (which is also divided by 2).
- Time Stamp Control: Enables or Disables Timestamp counter input to Nexus module
  - Free Running Counter (default)
  - Absolute Time stamp or relative time stamp modes
  - Start and stop on trigger events

Nexus Trace Buffer

The Nexus trace buffer consists of a 4096 Byte block of internal RAM (32 bits wide) used to capture Nexus message packets. This buffer can be used as an alternative way of using the Nexus trace capabilities through embedded software control. If the end user wishes to have some type of Nexus trace analysis or capture on the embedded platform itself then this buffer can be used for that purpose. This document only discusses Nexus Tracing through the CodeWarrior Tools. To program the Nexus module for embedded tracing, see the Argon+ or Argon LV L3 specifications and user guides.

Nexus Port FIFO

The Nexus Port also incorporates a 32 Message deep (message depth varies with message size and packing) FIFO. This FIFO can be set to stall the DSP or to suppress data messages and several depth settings.

3 Configuration of Nexus for Starcore/SDMA 1.X

The following section describes the CodeWarrior for StarCore and SDMA 1.x Nexus configuration panels and what each option does.

Nexus Configuration

Figure 2: Nexus Configuration shows the CodeWarrior for StarCore and SDMA 1.x tool's Nexus Configuration screen. This screen is primarily used to setup the Argon's Nexus Hardware interface so that it will function correctly with external trace capture hardware; typically this is an ARM RVT or a logic analyzer. The settings shown in Figure 2: Nexus Configuration below are typical settings for the ARM RVT with the DSP core clock speed set to 208 MHz. These settings can be interpreted as:

- Nexus Tracing is on (Enable Nexus)
- Nexus Port using 2 bit Message Start/End Output signaling (2-bit MSEO)
- Nexus Port clock = System clock /2 (Clock Divisor)
- Nexus Port is 16 bits wide (Port Mode)
- Post Trigger Fill is disabled
- Timestamps are disabled
- DDR mode not used
- Trigger position set to 0% (not used when Post Trigger Fill is disabled)
- Trace Buffer depth set to 2 million Nexus Messages (RVT buffer is 8Mbytes, Nexus messages average in size to about 4 bytes each)

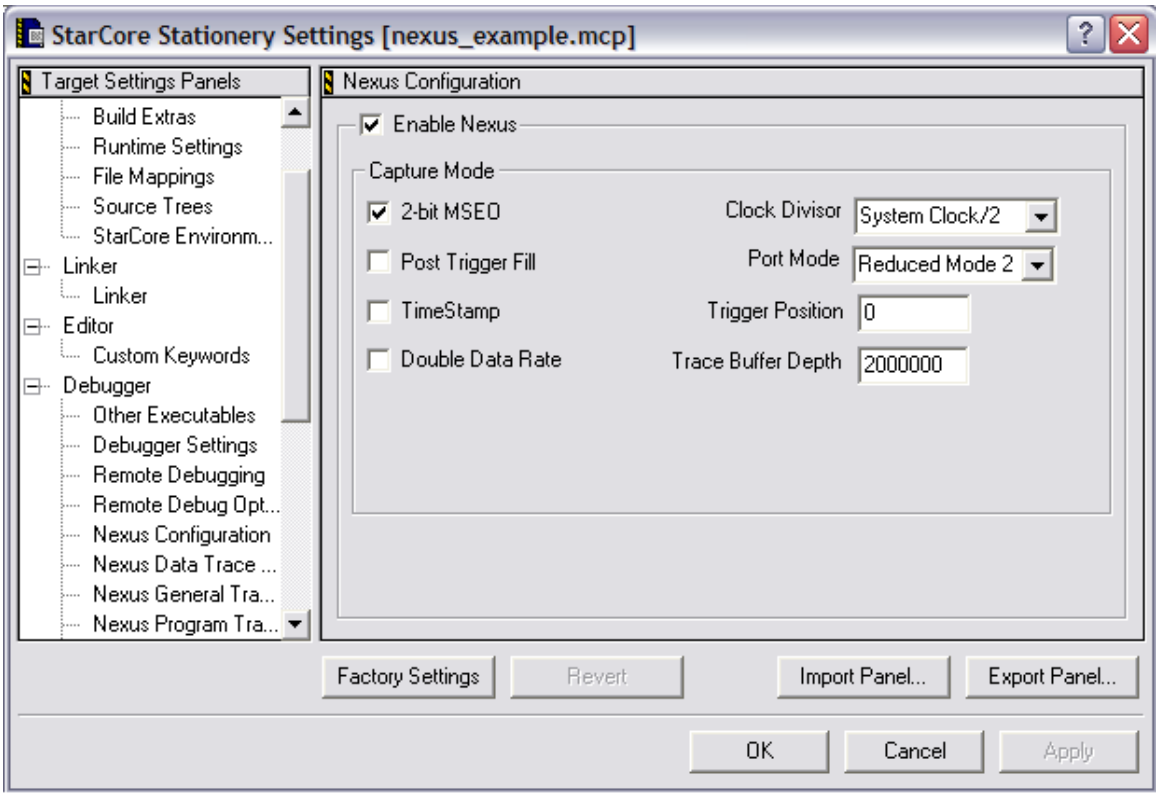

Figure 2: Nexus Configuration

Table 1: Nexus Configuration Options describes in more detail each of the available settings and how they can be interpreted.

Name	Settings	Description	Typical Setting with RVT
Enable Nexus	On/Off	Enable Nexus Hardware module	On
2-bit MSEO	On/Off	Enable 2 bit State machine. Allows faster message writes to Nexus Trace Buffer	On
Post Trigger Fill	On/Off	Enabling this causes the RVT trace buffer to continue to fill past the Trigger position setting until the trace buffer is full.	Off
Time Stamp	On/Off	Enables Timestamp input to Nexus Module	On if Time-stamps are required
Double Data Rate	On/Off	Divides Nexus Data clock by 2 and sends data out on both edges of divided clock	Off
Clock Divisor	SysClk SysClk/2 SysClk/4 SysClk/8	Setting to clock Nexus data output to trace hardware.	SysClk/2[1]
Port Mode	Reduced Mode 1 Reduced Mode 2	Set 8 bit message port width or Set 16 bit message port width[2].	Reduced Mode 2
Trigger Position	0 to 100 %	Sets the depth at which the trigger point in the RVT trace buffer will run. (50% implies triggering occurs when RVT trace buffer is 50% full). Enabling this with post trigger Fill enabled will capture trace before and after a trigger point.	Up to User
Trace Buffer Depth	0 to 2000000	RVT buffer depth in terms of Nexus Messages, maximum of 2 million messages[3].	2,000,000 or less

Table 1: Nexus Configuration Options

1. This setting should maintain the Nexus port at an equal or slower rate than the StarCore clock rate. If System clock is running at 416 MHz and StarCore is running at 208 MHz then clock divisor should be System Clock/2.
2. ARM RVT only supports up to a 16 bit trace interface.
3. Nexus Trace retrieval time increases proportionally to Trace Buffer depth

Nexus Program Trace Configuration

Figure 3: Nexus Program Trace Configuration shows the Nexus Program Trace Configuration panel. This panel allows the user to configure program tracing on a triggered basis and to allow timestamps to occur on a triggered basis also. If this is not configured, the Nexus module will output all program trace messages to the trace buffer and quickly over run it! There are two program trace trigger pairs. The first uses EOnCE events to start and stop program tracing. These can be setup using the EOnCE configuration tool, and provide advanced triggering capabilities

based on program counter, counters, and data accesses. The second uses 4 External Watchpoints which can be configured through the Argon's embedded cross trigger module's channels (ECT) and/or Process ID comparisons. Additionally Time Stamps can be placed on specific trigger start and end events also configured through ECT.

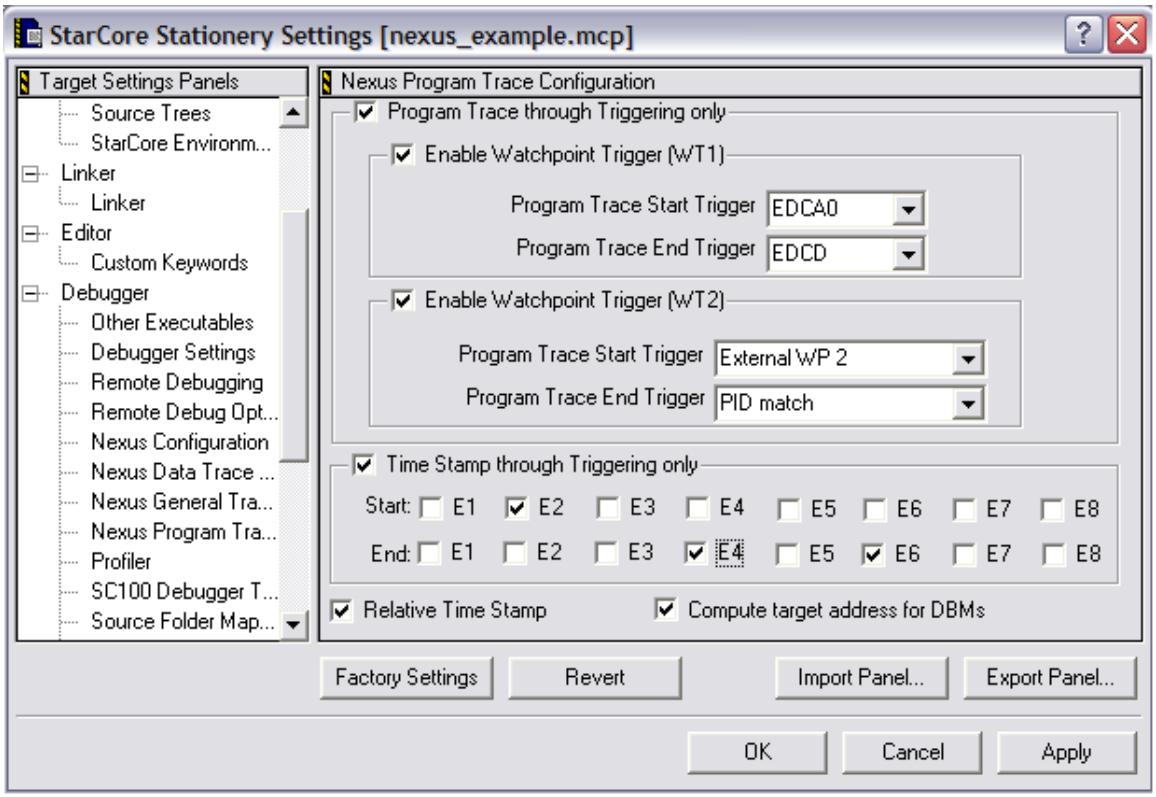

Figure 3: Nexus Program Trace Configuration

Table 2 Shows the Settings for the program tracing.

Name	Settings	Description
Program Trace through Triggering Only	On/Off	Turns Hardware trace triggering on and off. When this is on program tracing only
Enable Watchpoint Trigger WT1	On/Off	Enables triggering of Nexus Program Trace for EOnCE trigger events
Program Trace Start Trigger	None EDCA0 EDCA1 EDCA2 EDCA3 EDCA4 EDCA5 EDCD	Selects start of tracing as an EOnCE event
Program Trace End Trigger	None EDCA0 EDCA1 EDCA2 EDCA3 EDCA4 EDCA5 EDCD	Selects end of tracing as an EOnCE event
Enable Watchpoint Trigger WT2	On/Off	Enables triggering of Nexus Program Trace for ECT channel triggers and Process ID matching or mismatching
Program Trace Start Trigger	External WP1 External WP2 External WP3 External WP4 PID Match PID Mismatch	Selects start of tracing as an ECT channel 0 Selects start of tracing as an ECT channel 1 Selects start of tracing as an ECT channel 2 Selects start of tracing as an ECT channel 3 Selects start of tracing when PID or DID = set PID or DID (with Mask) Selects start of tracing when PID or DID != set PID or DID (with Mask)
Program Trace End Trigger	External WP1 External WP2 External WP3 External WP4 PID Match PID Mismatch	Selects end of tracing as an ECT channel 0 Selects end of tracing as an ECT channel 1 Selects end of tracing as an ECT channel 2 Selects end of tracing as an ECT channel 3 Selects end of tracing when PID or DID = set PID or DID (with Mask) Selects end of tracing when PID or DID != set PID or DID (with Mask)
Time Stamp through Triggering Only	On/Off	Turns timestamps on/off for program flow between ECT events only
Start E1	On/Off	This is an event trigger from the embedded cross trigger module. ECT SC140 CTI cti_trig_out[7]

Start E2	On/Off	This is an event trigger from the embedded cross trigger module. ECT SC140 CTI_cti_trig_out[6]
Start E3	On/Off	Not Connected on Argon+, Argon LV
Start E4	On/Off	Not Connected on Argon+, Argon LV
Start E5	On/Off	Not Connected on Argon+, Argon LV
Start E6	On/Off	Not Connected on Argon+, Argon LV
Start E7	On/Off	Not Connected on Argon+, Argon LV
Start E8	On/Off	Not Connected on Argon+, Argon LV
End E1	On/Off	This is an event trigger from the embedded cross trigger module. ECT SC140 CTI_cti_trig_out[7]
End E2	On/Off	This is an event trigger from the embedded cross trigger module. ECT SC140 CTI_cti_trig_out[6]
End E3	On/Off	Not Connected on Argon+, Argon LV
End E4	On/Off	Not Connected on Argon+, Argon LV
End E5	On/Off	Not Connected on Argon+, Argon LV
End E6	On/Off	Not Connected on Argon+, Argon LV
End E7	On/Off	Not Connected on Argon+, Argon LV
End E8	On/Off	Not Connected on Argon+, Argon LV
Relative Time Stamp	On/Off	Changes Time stamps from absolute values to relative values
Compute target address for DBMs	On/Off	Enables address decoding by CodeWarrior for Direct Branch Messages[2]

Table 2: Nexus Program Trace Options

1. Please note that ECT Channel mappings range from 0 to 3 on the ECT module, which correspond to Nexus External WP1 to 4 on the Nexus module.
2. This feature will likely be removed in future tool releases and translation will always be enabled.

Nexus Data Trace Configuration

Figure 4: Nexus Data Trace Configuration shows the Nexus Data Trace Configuration panel. This panel is used to configure four hardware data trace modules and to use two hardware Data trace triggers to set triggers to limit data tracing to a start and end time, with out these triggers data trace messages will be generated for all data trace accesses defined in the control panel.

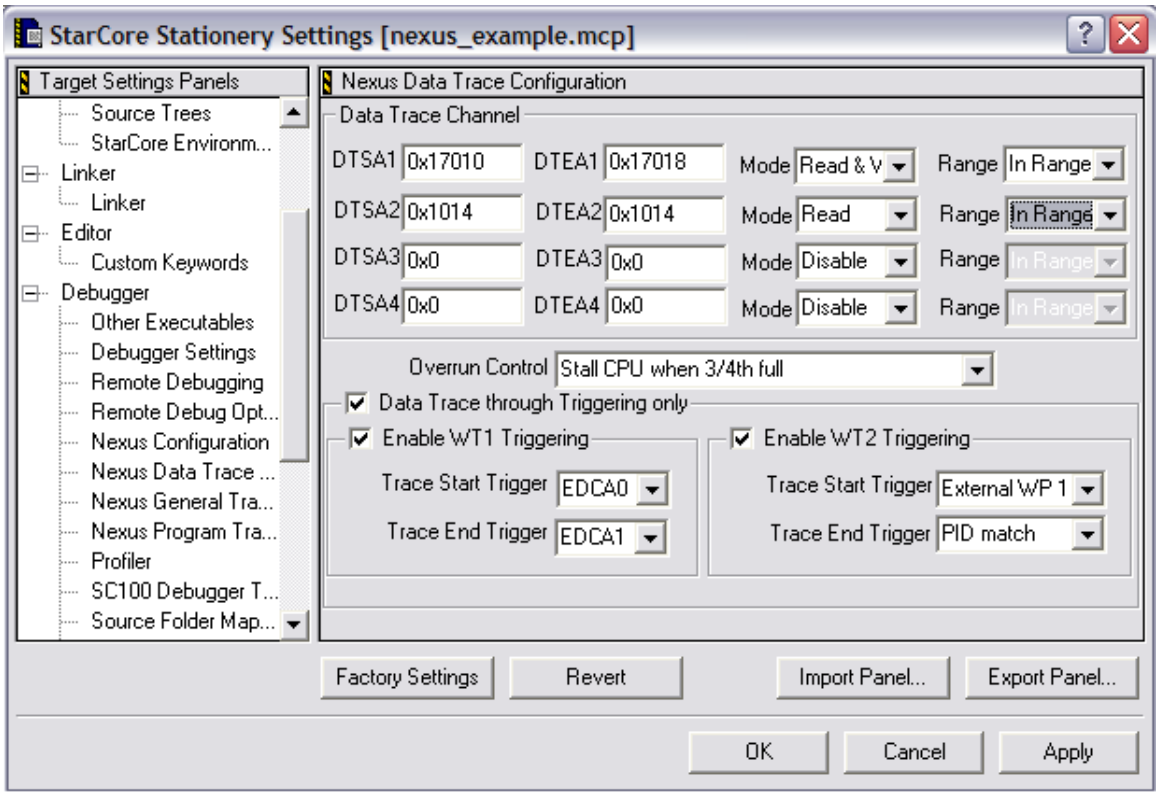

Figure 4: Nexus Data Trace Configuration

Table 3: Nexus Data Trace Configuration Options gives details on the Nexus Data Trace configuration options which are available.

Name	Settings	Description
DTSA1	32 bit Address	Start Address for Data trace channel
DTEA1	32 bit Address	End Address for Data trace channel. End Address must be greater than or equal to start address or no tracing will occur.
Mode	Disabled Read Write Read and Write	Data tracing is disabled Data tracing is for Read access Data tracing is for Write access Data tracing is for Read/Write access
Range	In Range Outside Range	In Range will trace all data from Address = DTSA1 to Address = DTEA1 Outside Range will trace all data access excluding Address= DTSA1 to Address = DTEA1
DTSA2	32 bit Address	Start Address for Data trace channel
DTEA2	32 bit Address	End Address for Data trace channel. End Address must be greater than or equal to start address or no tracing will occur.
Mode	Disabled Read Write Read and Write	Data tracing is disabled Data tracing is for Read access Data tracing is for Write access Data tracing is for Read/Write access
Range	In Range Outside Range	In Range will trace all data from Address = DTSA2 to Address = DTEA2 Outside Range will trace all data access excluding Address= DTSA2 to Address = DTEA2
DTSA3	32 bit Address	Start Address for Data trace channel
DTEA3	32 bit Address	End Address for Data trace channel. End Address must be greater than or equal to start address or no tracing will occur.
Mode	Disabled Read Write Read and Write	Data tracing is disabled Data tracing is for Read access Data tracing is for Write access Data tracing is for Read/Write access
Range	In Range Outside Range	In Range will trace all data from Address = DTSA3 to Address = DTEA3 Outside Range will trace all data access excluding Address= DTSA3 to Address = DTEA3
DTSA4	32 bit Address	Start Address for Data trace channel
DTEA4	32 bit Address	End Address for Data trace channel. End Address must be greater than or equal to start address or no tracing will occur.
Mode	Disabled Read Write Read and Write	Data tracing is disabled Data tracing is for Read access Data tracing is for Write access Data tracing is for Read/Write access

Range	In Range Outside Range	In Range will trace all data from Address = DTSA4 to Address = DTEA4 Outside Range will trace all data access excluding Address= DTSA4 to Address = DTEA4
Overrun Control	No Stall or Suppression Stall CPU when ¼ full Stall CPU when ½ full Stall CPU when ¾ full Suppress Trace when ¼ full Suppress Trace when ½ full Suppress Trace when ¾ full	Options for holding off data tracing by delaying the CPU or ending the data tracing at specific thresholds of the Nexus port's output FIFO . Setting to no Stall or Suppression will allow the FIFO to free run.[1]
Data Trace through Triggering Only	On/Off	Allows Data Tracing to start and end on specific trigger events such as EOnCE events or hardware Watchpoints
Enable WT1	On/Off	Enables triggering of Nexus Data Trace for EOnCE trigger events
Trace Start Trigger	None EDCA0 EDCA1 EDCA2 EDCA3 EDCA4 EDCA5 EDCD	Selects start of tracing EOnCE event
Trace End Trigger	None EDCA0 EDCA1 EDCA2 EDCA3 EDCA4 EDCA5 EDCD	Selects end of tracing EOnCE event
Enable WT2	On/Off	Enables triggering of Nexus Program Trace for ECT channel triggers and Process ID matching or mismatching
Trace Start Trigger	External WP1 External WP2 External WP3 External WP4 PID Match PID Mismatch	Selects start of tracing as an ECT channel 0 Selects start of tracing as an ECT channel 1 Selects start of tracing as an ECT channel 2 Selects start of tracing as an ECT channel 3 Selects start of tracing when PID or DID = set PID or DID (with Mask) Selects start of tracing when PID or DID != set PID or DID (with Mask)

Trace End Trigger	External WP1	Selects end of tracing as an ECT channel 0
	External WP2	Selects end of tracing as an ECT channel 1
	External WP3	Selects end of tracing as an ECT channel 2
	External WP4	Selects end of tracing as an ECT channel 3
	PID Match	Selects end of tracing when PID or DID = set PID or DID (with Mask)
	PID Mismatch	Selects end of tracing when PID or DID != set PID or DID (with Mask)

Table 3: Nexus Data Trace Configuration Options

1. This feature is used when data tracing may create large amounts of data which are placed into the output FIFO faster than the connecting hardware (such as an ARM RVT) can read the data out.

Nexus General Trace Configuration

The General Trace Configuration Panel (see Figure 5: Nexus General Trace Configuration) is used to control additional features of the Nexus hardware and port interface. These include Nexus Output event generation to an output pin, Nexus Input Event control, Data Acquisition, and Process ID based tracing settings.

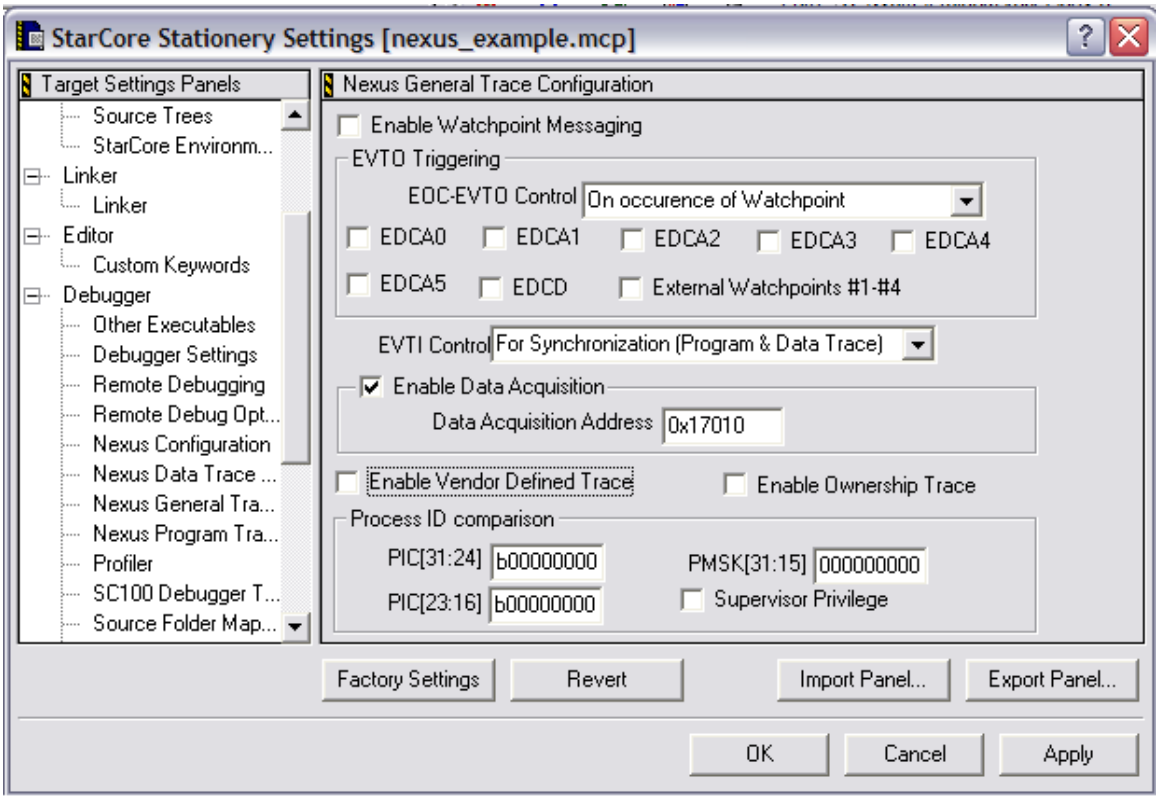

Figure 5: Nexus General Trace Configuration

Table 4 Shows details on each of the controls for the Nexus General Trace Configuration panel.

Name	Settings	Description
Enable Watchpoint Messaging	On/Off	Turns On/Off Nexus Message generation when a watchpoint is hit
EOC-EVTO Control	On Occurrence of Watchpoint On Entry into System Level Debug Mode	Event Trigger Output occurs when below events occur. Event Trigger Output occurs when Starcore enters debug mode
EDCA0	On/Off	Enables EVTO toggle around EDCA0
EDCA1	On/Off	Enables EVTO toggle around EDCA1
EDCA2	On/Off	Enables EVTO toggle around EDCA2
EDCA3	On/Off	Enables EVTO toggle around EDCA3
EDCA4	On/Off	Enables EVTO toggle around EDCA4
EDCA5	On/Off	Enables EVTO toggle around EDCA5
EDCD	On/Off	Enables EVTO toggle around EDCD
External Watchpoints 1-4	On/Off	Enables EVTO toggle around ECT channel mapped trigger signals
EVTI Control	For Synchronization (Program & Data Trace) Assert debug request to processor Disabled	Event Trigger Input: Causes next Program or Data Trace message to be a Synchronization message[1] when EVTI pin is toggled high to low Halts processor when EVTI pin is toggled high to low No EVTI pin control
Enable Data Acquisition	On/Off	
Data Acquisition Address	32 bit Address	Address to Trace Data Writes
Enable Vendor Defined Trace	On/Off	Creates Nexus Messages containing DPU counter data[2]
Enable Ownership Trace	On/Off	Sends a Nexus message whenever PID or Data device ID changes
PIC[31:24]	8 bit value	Process ID mask
PIC[23:16]	8 bit value	Data ID mask
PMSK[31:15]	17 bit value	Masks out which bits in PID|DID|Supervisor (PIC[31:24] | PIC[23:16] | Supervisor bit) bits are being used for comparisons (eg all '1's will cause an exact comparison)[3]
Supervisor Privilege	On/Off	Single Bit for comparisons in Supervisor mode(on) or User Mode (off)

Table 4: Nexus General Trace Options

1. *Synchronous messages contain a full address; subsequent messages only contain addresses with bits which have changed.*
2. *DPU counters are configured in profiler options*

3. *Process ID comparisons can be configured with a PID, DID and Supervisor/User bit wise settings. Additionally each of these settings can have a specific bit mask to mask on or off the comparisons of the PID, DID and Supervisor/User settings. For Example: PID = 0xF0, DID = 0x 0F, S/U = 0x1. To see all PID matches for the PID and DID settings but to see them for Supervisor mode and User mode: PMSK = PID | DID | S/U = 0x1111111111111110b or 0x1FFFE. This causes the PID and DID comparisons to occur, but the S/U bit comparison will not be used.*

4 Configuration of EOnCE

The EOnCE module provides dedicated hardware for generating real time signaling of events which occur while code is executing. These events can be used by the Nexus module to trigger tracing on and off, and to setup more advanced sequenced triggering configurations for program flow and data tracing. The EOnCE Data and Address Detection channels are used to configure hardware breakpoints and hardware watchpoints. These are used as shared resources by the debugger. For example, if the user assigns three hardware breakpoints, then these three addresses will be configured on the EOnCE EDCA0, EDCA1, and EDCA2 detection channels (provided these are the first three HW Breakpoints or Watchpoints being set).

EE Pins Controller (Control)

The EOnCE control tab (see Figure 6: EOnCE Control Settings) is used to configure the EE pin signals used by the embedded cross trigger (ECT). These signals can be routed into and out of the ECT to other hardware modules on the Argon platform. A total of 2 pins are configurable as outputs on the DSP's ECT. These signals can be routed to other ECT inputs on the ARM, SDMA, or Platform ECT interfaces. EE Pin 1 through 5 are not available on the Argon platform and are present in the tool for future hardware designs which may implement these signals. The EE pin signals are described in Table 5: EOnCE Control Settings.

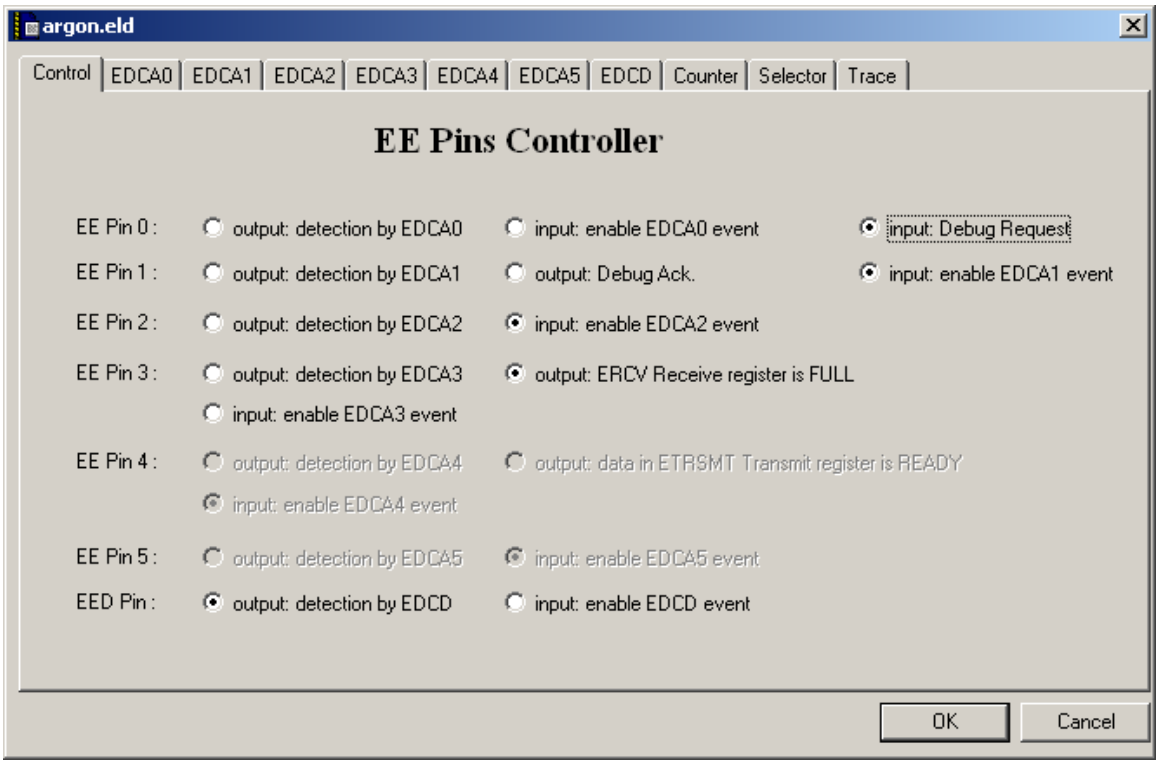

Figure 6: EOnCE Control Settings

Name	Settings	Description
EE Pin 0	output: detection by EDCA0 input: enable EDCA0 input: debug request	Not Connected Input from ECT (sin_ee_in[0]) enables EDCA0 event on rising edge Not Connected
EE Pin 1	output: detection by EDCA1 output: debug ack input: enable EDCA1	Not Connected Not Connected Not Connected
EE Pin 2	output: detection by EDCA2 input: enable EDCA2	Not Connected Not Connected
EE Pin 3	output: detection by EDCA3 output: ERCV receive register full input: enable EDCA3	Not Connected Not Connected Not Connected
EE Pin 4	output: detection by EDCA4 input: enable EDCA4	Not Connected Not Connected
EE Pin 5	output: detection by EDCA5 input: enable EDCA5	Not Connected Not Connected
EED Pin	output: detection by EDCD input: enable EDCD	Not Connected Input from ECT (sin_eed_in) enables EDCD on rising edge

Table 5: EOnCE Control Settings

Address Event Detection Channel (EDCA0 through 5)

The EOnCE module supports 6 address detection units for generating address based events. These events can be generated to cover a single address, an address range or exclude either.

Address events can be on data address bus A, data address bus B, both data address busses A and B, or based on the program counter (instruction bus). Additionally these events can be configured to only trigger after prior events or signals have occurred. Figure 7: EOnCE Address Event Detection Channel Settings shows the configuration panel for EDCA1 which is identical to the other 5 configuration panels for Address Event Detection Channels.

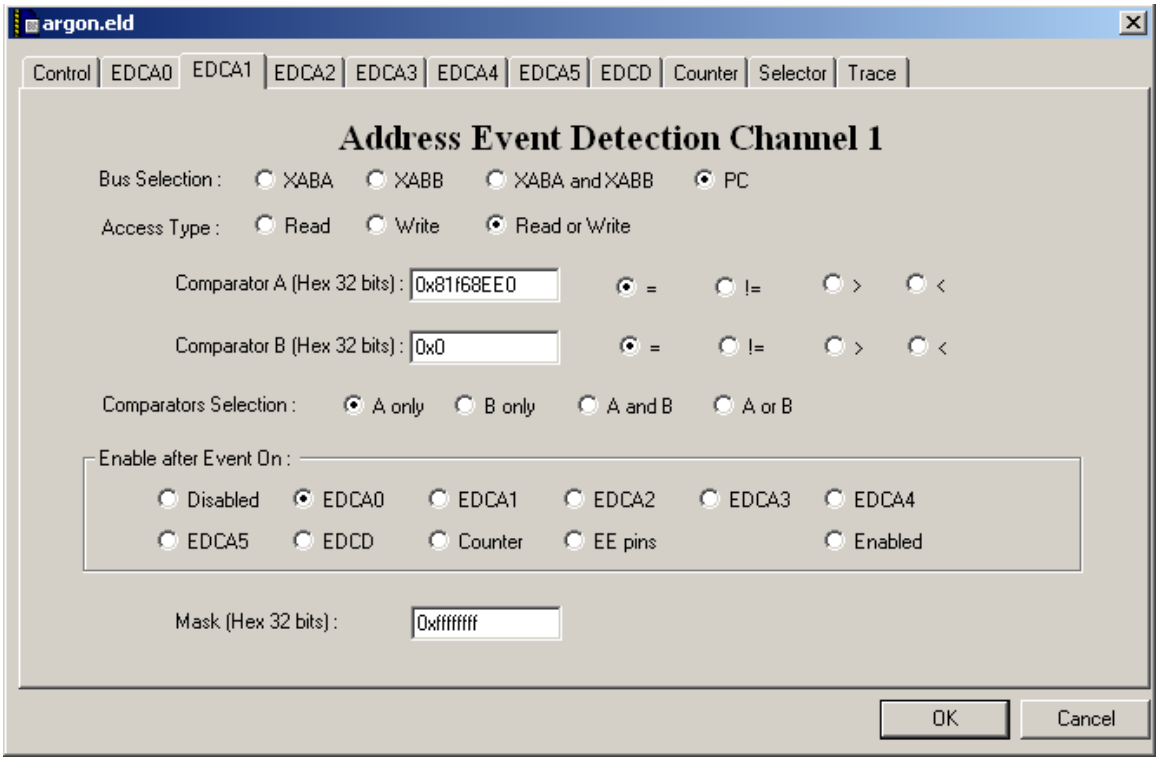

Figure 7: EOnCE Address Event Detection Channel Settings

Table 6: EOnCE Event Detection Configuration Settings shows the definitions for Address Event Detection Channel settings.

Name	Settings	Description
Bus Selection	XABA XABB XABA and XABB PC	Event detection on data bus A Event detection on data bus B Event detection on data bus A or B[1] Event detection on Program Counter
Access Type	Read Write Read or Write	Detect on Read access Detect on Write access Detect on Read or Write access
Comparator A	32 bit Value = != > <	Value to compare with bus Bus equals value Bus not equals value Bus greater than value Bus less than value
Comparator B	32 bit Value = != > <	Value to compare with bus Bus equals value Bus not equals value Bus greater than value Bus less than value
Comparator Selection	A only B only A and B A or B	Use Comparator A Use Comparator B Use logical AND of Comparator A and Comparator B Use logical OR of Comparator A and Comparator B
Enable After Event On[2]	Disabled EDCA0 EDCA1 EDCA2 EDCA3 EDCA4 EDCA5 EDCD Counter EE pins Enabled	Disable this event Trigger this event after EDCA0 occurs Trigger this event after EDCA1 occurs Trigger this event after EDCA2 occurs Trigger this event after EDCA3 occurs Trigger this event after EDCA4 occurs Trigger this event after EDCA5 occurs Trigger this event after EDCD occurs Trigger this event after counter decrements past 0 Trigger this event after ee pin 0 signal Always enable this event
Mask	32 bit value	Mask the comparison values with this bit mask[3]

Table 6: EOnCE Event Detection Configuration Settings

1. A and B mode performs a comparison of data address bus A on Comparator A and data address bus B on Comparator B
2. Note that enabling EDCA(N) after Event on EDCA(N) is not a valid configuration though the tool exposes it in the EDCA interface (eg EDCA4 enabled after event on EDCA4).
3. Address Masking will automatically ignore the lsb(s) for WORD and LONG access types

Data Event Detection Channel (EDCD)

The EOnCE module has one Data Event channel which can be configured to compare data reads or data writes of different access sizes to a specific value (which can also be bit masked). This event can be enabled by the occurrence of other events or be configured to always be enabled.

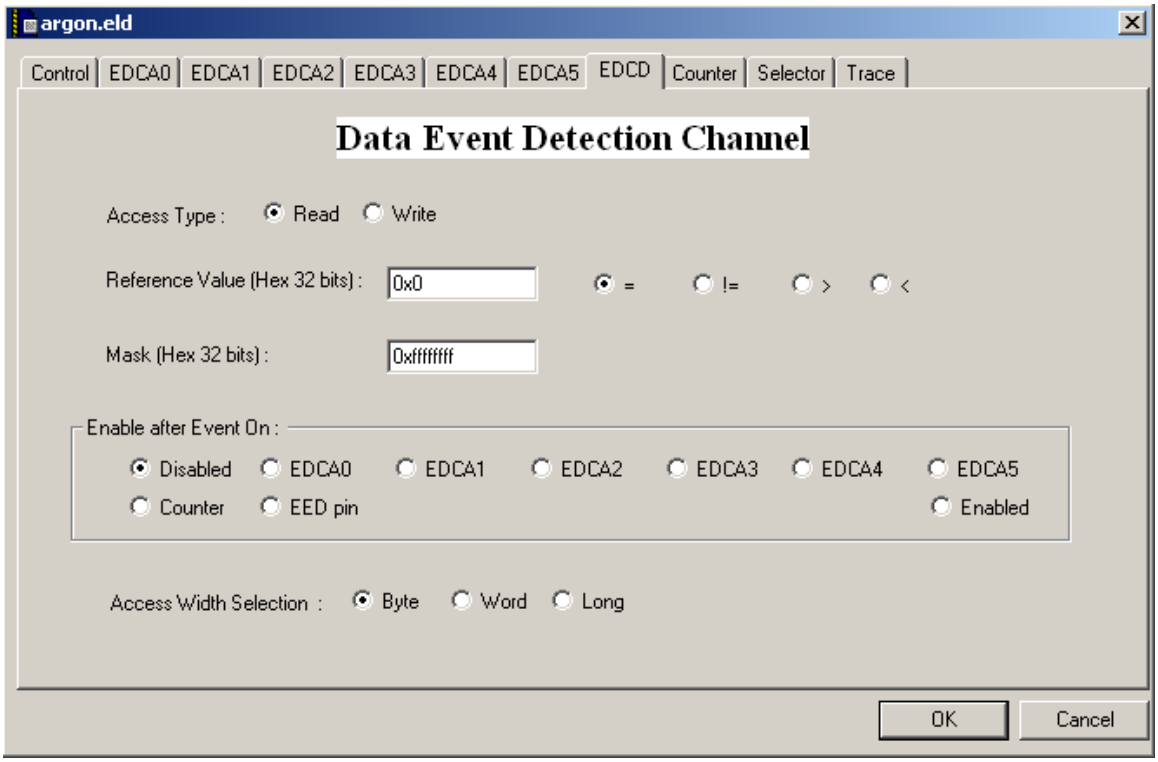

Figure 8: EOnCE Data Event Detection Channel Settings

Name	Settings	Description
Access Type	Read Write	Trigger on Read access Trigger on Write access
Reference Value	32 bit Value = != > <	Value to compare with bus Bus equals value Bus not equals value Bus greater than value Bus less than value
Mask	32 bit value	Value to bit Mask reference value (set bits are used, cleared bits are not)
Enable After Event On	Disabled EDCA0 EDCA1 EDCA2 EDCA3 EDCA4 EDCA5 Counter EED pin Enabled	Disable this event Trigger this event after EDCA0 occurs Trigger this event after EDCA1 occurs Trigger this event after EDCA2 occurs Trigger this event after EDCA3 occurs Trigger this event after EDCA4 occurs Trigger this event after EDCA5 occurs Trigger this event after counter decrements past 0 Trigger this event after EED pin signal Always enable this event
Access Width Selection	Byte Word Long	8 bit accesses 16 bit accesses 32 bit accesses

Table 7: EOnCE Data Event Detection Settings

Event Counter (Counter)

The EOnCE Counter allows for EOnCE Address channel events, EOnCE data channel events, clock cycles, DEBUGEV instructions, execution sets and trace events to be counted before enabling other events. This allows for advanced triggering options such as triggering after multiple instances of an event occur, a specific number of clock cycles (specific time), DEBUGEV instructions, trace events, or execution sets. Figure 9: EOnCE Counter Settings shows the Counter settings panel.

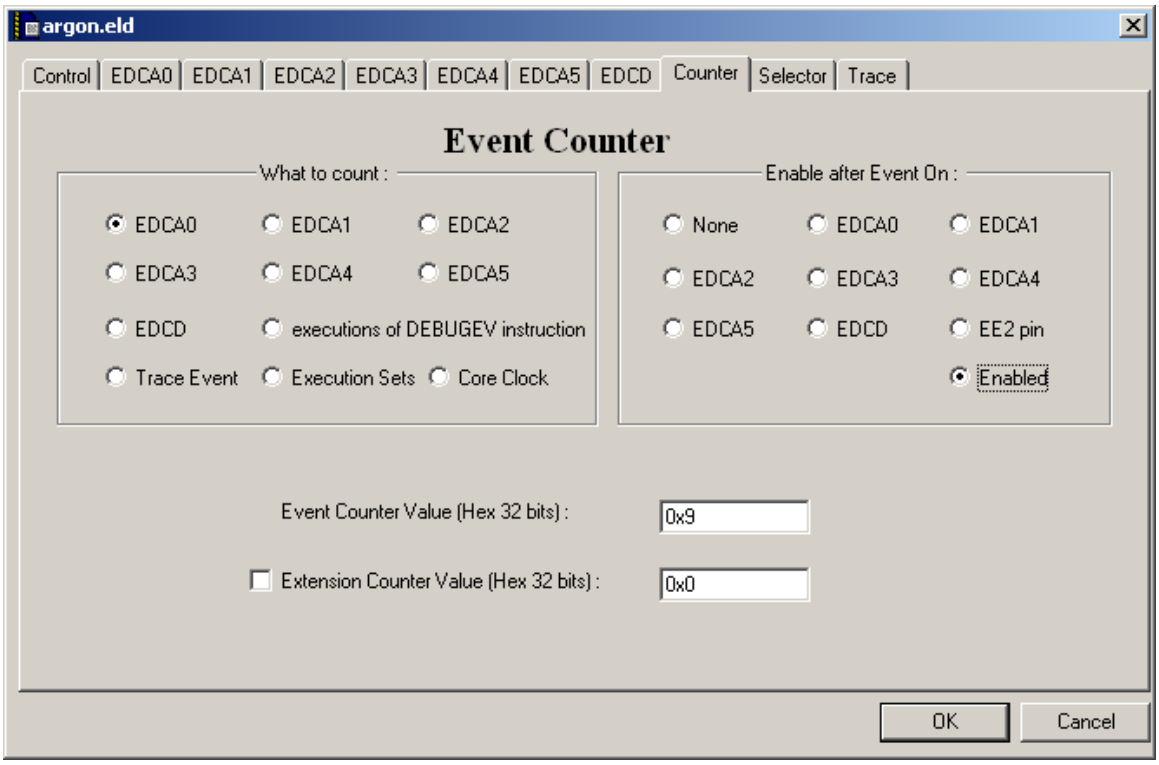

Figure 9: EOnCE Counter Settings

Table 8: EOnCE Counter Settings shows the definition of all of the counter settings for the EOnCE counter.

Name	Settings	Description
What to Count	EDCA0	Count EDCA0 events
	EDCA1	Count EDCA1 events
	EDCA2	Count EDCA2 events
	EDCA3	Count EDCA3 events
	EDCA4	Count EDCA4 events
	EDCA5	Count EDCA5 events
	EDCD	Count EDCD events
	DEBUGEV instructions	Software Breakpoints
	Trace Event	EOnCE Trace Event
	Execution Sets	Number of execution sets
	Core Clock	# of Clock ticks
Enable after Event On	None	Counter is disabled
	EDCA0	Counter enabled after EDCA0 event
	EDCA1	Counter enabled after EDCA1 event
	EDCA2	Counter enabled after EDCA2 event
	EDCA3	Counter enabled after EDCA3 event
	EDCA4	Counter enabled after EDCA4 event
	EDCA5	Counter enabled after EDCA5 event
	EDCD	Counter enabled after EDCD event
	EE2 Pin	Counter enabled after EE2 pin toggle
	Enabled	Counter always enabled to count
Event Counter Value	31 bit value	Value to count. Counter triggers when count decrements past 0 so a value of 9 will trigger the counter event on the 10th instance of “What to Count”
Extension Counter Value	31 bit value	Extended count to increase the range of the counter

Table 8: EOnCE Counter Settings

The counter will count “What to Count” and will trigger a counter event on the (Event Counter + 1)th time the “What to Count” event occurs. The counter event can be used to enable other EOnCE events. It can also be enabled (started) by other EOnCE events as well.

Event Selector (Selector)

The event selector tab (see Figure 10: EOnCE Event Selector Settings) allows for EOnCE events to trigger four different actions: Entry to Debug Mode (halts processor with a HW BP), cause a Debug Exception to occur, cause tracing to virtual trace buffer to start, and cause tracing to end. Each of the four cases can logically AND or logically OR the events so a combination of events is required for the action to be taken or multiple events can cause the action to be taken.

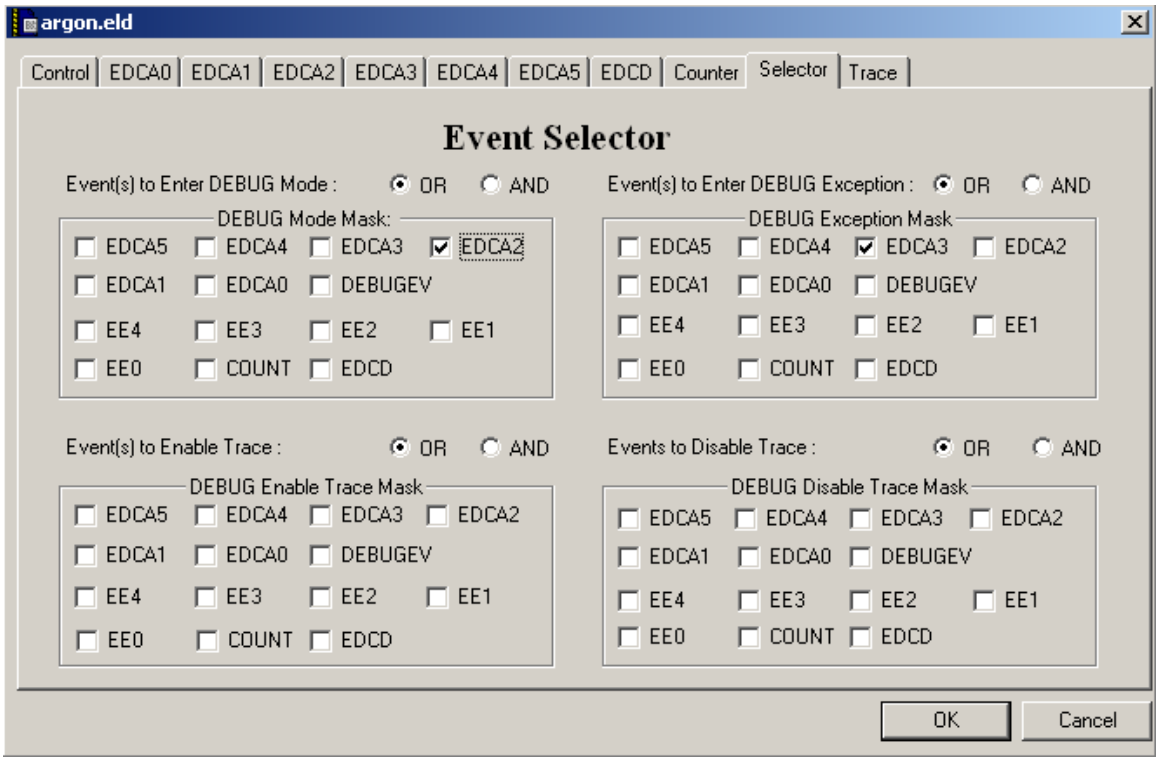

Figure 10: EOnCE Event Selector Settings

Table 9: EOnCE Selector Options shows the settings available for the Event Selector.

Name	Settings	Description
Event to Enter Debug mode[1]		Halts StarCore on:
	EDCA5	Address Channel 5 Event
	EDCA4	Address Channel 4 Event
	EDCA3	Address Channel 3 Event
	EDCA2	Address Channel 2 Event
	EDCA1	Address Channel 1 Event
	EDCA0	Address Channel 0 Event
	DEBUGEV	DEBUGEV Instruction occurs
	EE4	EE pin 4 Event
	EE3	EE pin 3 Event
	EE2	EE pin 2 Event
	EE1	EE pin 1 Event
	EE0	EE pin 0 Event
	COUNT	Counter Event
	EDCD	Data channel detection Event
Event to Enter Debug Exception		Causes Debug exception on:
	EDCA5	Address Channel 5 Event
	EDCA4	Address Channel 4 Event
	EDCA3	Address Channel 3 Event
	EDCA2	Address Channel 2 Event
	EDCA1	Address Channel 1 Event
	EDCA0	Address Channel 0 Event
	DEBUGEV	DEBUGEV Instruction occurs
	EE4	EE pin 4 Event
	EE3	EE pin 3 Event
	EE2	EE pin 2 Event
	EE1	EE pin 1 Event
	EE0	EE pin 0 Event
	COUNT	Counter Event
	EDCD	Data channel detection Event
Event to Enable Trace Mask		Enables EOnCE (or VTB) Tracing on:
	EDCA5	Address Channel 5 Event
	EDCA4	Address Channel 4 Event
	EDCA3	Address Channel 3 Event
	EDCA2	Address Channel 2 Event
	EDCA1	Address Channel 1 Event
	EDCA0	Address Channel 0 Event
	DEBUGEV	DEBUGEV Instruction occurs
	EE4	EE pin 4 Event
	EE3	EE pin 3 Event
	EE2	EE pin 2 Event
	EE1	EE pin 1 Event
	EE0	EE pin 0 Event
	COUNT	Counter Event
	EDCD	Data channel detection Event

Event to Disable Trace Mask		Disables EOnCE (or VTB) Tracing on:
	EDCA5	Address Channel 5 Event
	EDCA4	Address Channel 4 Event
	EDCA3	Address Channel 3 Event
	EDCA2	Address Channel 2 Event
	EDCA1	Address Channel 1 Event
	EDCA0	Address Channel 0 Event
	DEBUGEV	DEBUGEV Instruction occurs
	EE4	EE pin 4 Event
	EE3	EE pin 3 Event
	EE2	EE pin 2 Event
	EE1	EE pin 1 Event
	EE0	EE pin 0 Event
	COUNT	Counter Event
	EDCD	Data channel detection Event

Table 9: EOnCE Selector Options

1. Entering Debug state causes a Hardware breakpoint to occur. On the StarCore processor a Hardware breakpoint can be subject to "skidding", meaning it can miss (run past) the actual assembly opcode by up to 3 instructions.

Trace Unit (Trace)

The EOnCE Trace configuration sets up the type of trace data and control for EOnCE tracing which can be piped to the VTB (virtual trace buffer). VTB tracing is configured in the project settings panel, and in the project itself in the form of reserved memory (RAM) and an ISR routine. All VTB trace data comes from the EOnCE trace unit. Nexus trace is independent of this trace unit.

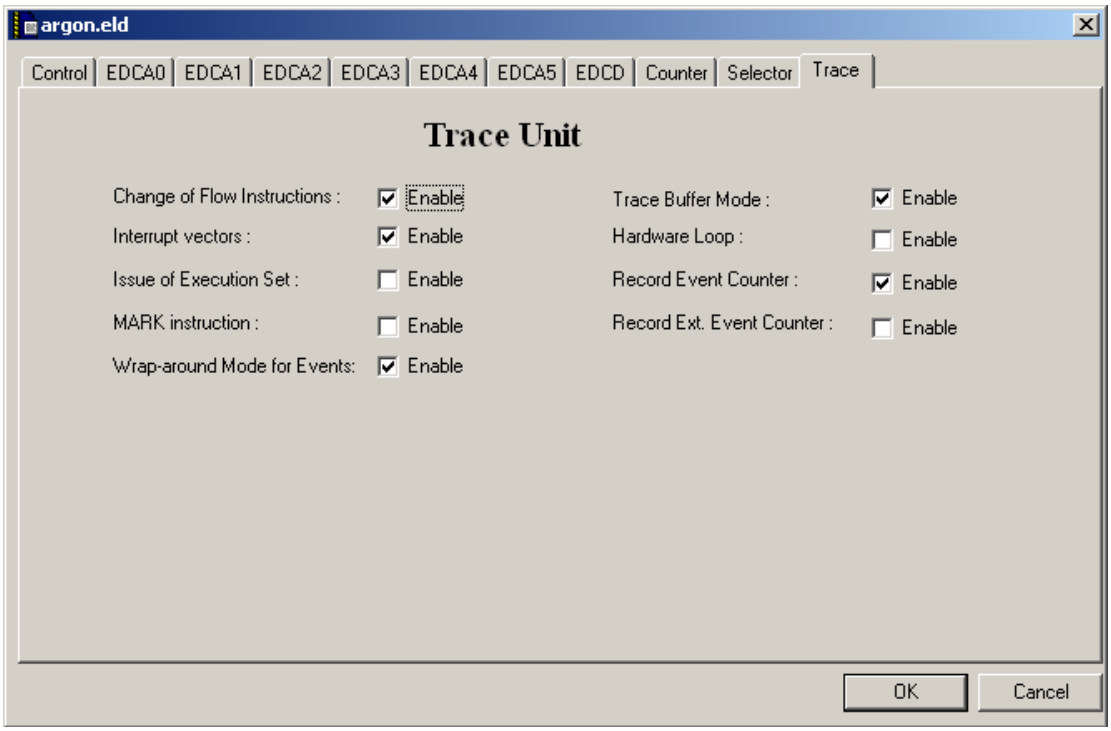

Figure 11: EOnCE Trace Configuration

Name	Settings	Description
Change of Flow instructions	On/Off	EOnCE tracing shows Change of flow instructions
Interrupt Vectors	On/Off	EOnCE tracing shows Interrupt vectors being taken
Issue of Execution Set[1]	On/Off	EOnCE tracing shows execution sets occurring
MARK instruction[2]	On/Off	EOnCE tracing shows MARK instruction
Wrap around Mode for Events	On/Off	EOnCE tracing will write in a circular buffer fashion
Trace Buffer Mode[3]	On/Off	Turns on EOnCE Tracing
Hardware Loop	On/Off	EOnCE tracing shows hardware loops occurring
Record Event Counter	On/Off	EOnCE tracing shows event counter
Record Ext. Event Counter	On/Off	EOnCE tracing shows extended event counter

Table 10: EOnCE Trace Buffer Settings

1. Adding this to the EOnCE Trace will preclude adding other trace data (Change of Flow, Interrupt Vectors, MARK instruction, Record Event Counter, Record Extended Event Counter)
2. Adding this to the EOnCE Trace will preclude adding other trace data (Change of Flow, Interrupt Vectors, Issue of Execution Set, Record Event Counter, Record Extended Event Counter)
3. Enabling VTB here will require enabling VTB in the project settings panel.

5 Nexus Trace Examples

The following chapter gives several use cases for Nexus Tracing. Each use case is setup as follows:

- Problem being investigated
- Initial Tool setup
- Execution and capturing of the data

Program Trace: Capture an Exception

In an embedded development environment a software exception (or panic) can occur and cause the embedded processor to run to an exception handler and spin infinitely, halting normal function of the system. It is often difficult after the exception has occurred to determine what happened in the embedded system to cause the exception. This case shows how to use Nexus Trace to view the program flow of events prior to running into an exception handler.

What is needed:

- CodeWarrior for StarCore/SDMA 1.0.3 or later software
- ARM RVI (Real View ICE)+ARM RVT (Real View Trace) hardware
- ARM RVT Ribbon to Mictor connector for ETM and Nexus (this is s special connector for use with a Nexus port)
- Argon + or Argon LV Hardware platform with JTAG port and 16 bit Nexus trace port on Mictor connector
- Software Project including
  - ELF debug file
  - Linker generated map file
  - source code (optional)

Steps to capture an exception trace:

1.) Load the project

   a. Configuration of EOnCE is performed real-time while connected to the hardware so a debug connection is necessary.
   b. Start the debug session, halting at the entry point will allow configuration of EOnCE to be used for this debug session without missing program execution.
   c. If there is a prior saved EOnCE configuration it will be used as the default configuration for EOnCE

2. Setup EOnCE events
   a. Setup EOnCE Event to be used to trigger end of tracing (see Figure 12: Trigger Setup for End of Trace Collection)

i. Find a point in the source code which is inside the exception handler of interest.
ii. Using the project's map file search for the program address of a point in the source code inside the exception handler. An alternate method would be to execute the program until the exception is reached, in the debug view switch the source view to mixed or assembly view and note the address inside the exception handler.
iii. In the EOnCE configuration window select the EDCA1 tab
iv. Set Bus Selection to "PC"
v. Set Access Type to "Read or Write"
vi. Set Comparator A to equal Address inside exception handler
vii. Set Comparators selection to "A only"
viii. Set Enable after Event On to "Enabled"

1. Note since this is the end of trace trigger this event can be set to be enabled only after EDCA0 occurs by selecting "EDCA0" as shown in Figure 12: Trigger Setup for End of Trace Collection

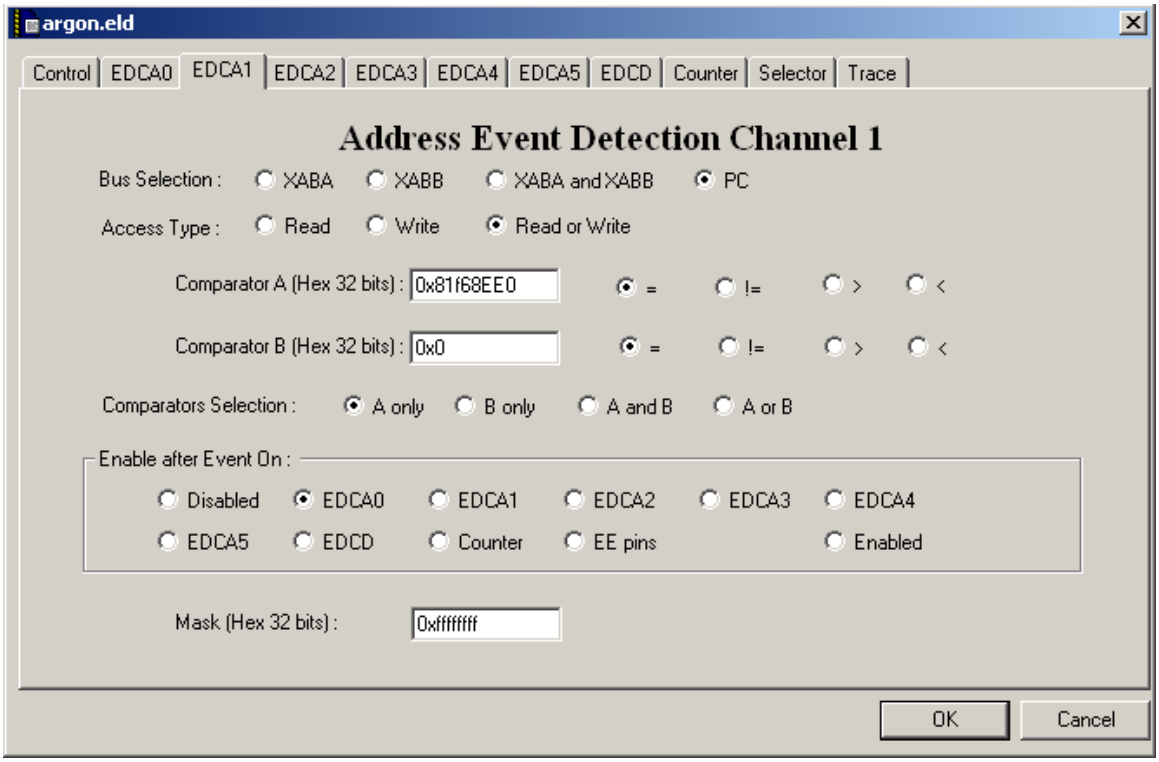

Figure 12: Trigger Setup for End of Trace Collection

b. Setup an EOnCE event to be used to trigger the start of tracing
i. Find a point in the source code which is after initialization code, and before any main execution or message loops. This should be a safe starting place where no exceptions are suspected to occur.
ii. In the project's map file search for the program address of a point in the source code which executes before any suspected problems where an exception would occur could be. If it is not known where the exception is occurring or an

exception occurs immediately upon execution this can be set to the program's entry point.

iii. In the EOnCE configuration window select the EDCA0 tab (see Figure 13: Trigger Setup for Start of Trace Collection)

iv. Set Bus selection to "PC"

v. Set Access Type to "Read or Write"

vi. Set Comparator A to the address where trace should begin

vii. Set Comparator Selection to "A only"

viii. Set Enable after Event On to "Enabled"

ix. Be sure EDCA0 is set to the enabled setting

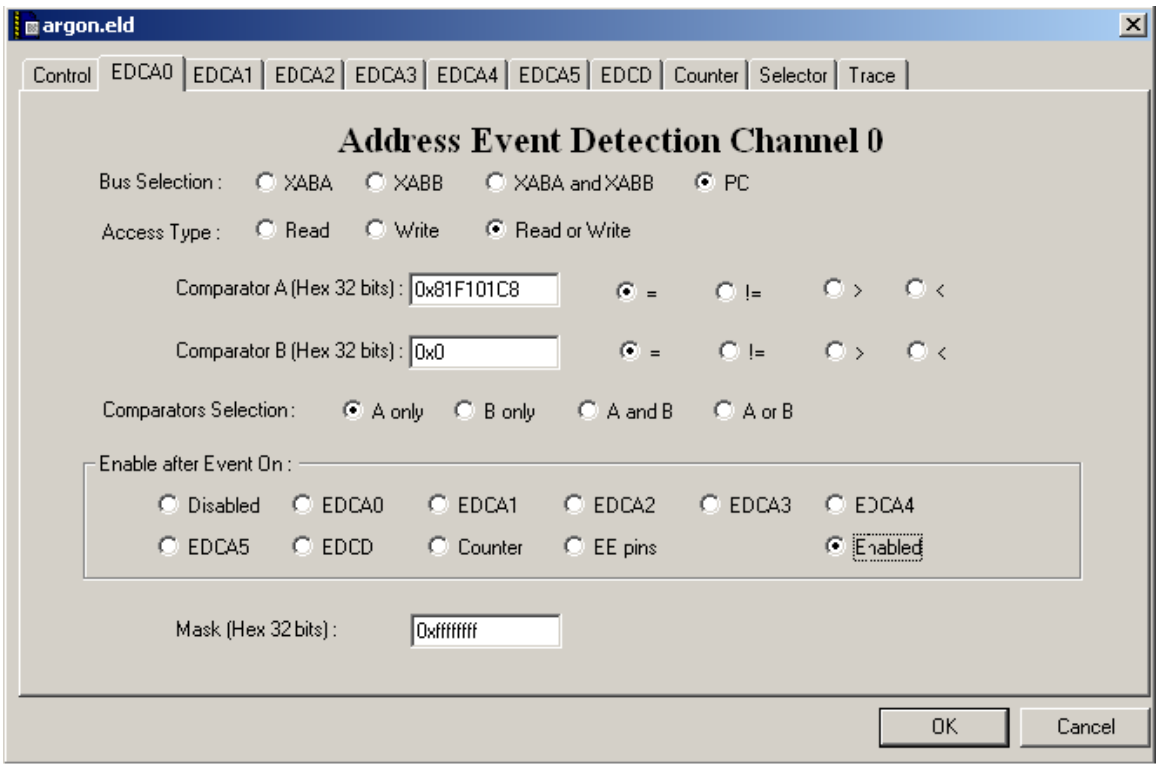

Figure 13: Trigger Setup for Start of Trace Collection

c. Setup EOnCE event to halt StarCore processor on occurrence (Figure 14: EOnCE Event Selector)

i. In the EOnCE configuration menu select the Selector Tab

ii. In the box labeled "Event(s) to Enter DEBUG Mode:"

1. Set the mode to "OR"
2. Select EDCA1

iii. This will cause the StarCore processor to halt when the EDCA1 (exception) event occurs, this is useful to indicate to the user that the exception was hit

iv. Other menu options provided here give similar control to the Nexus Configuration for enabling and disabling trace

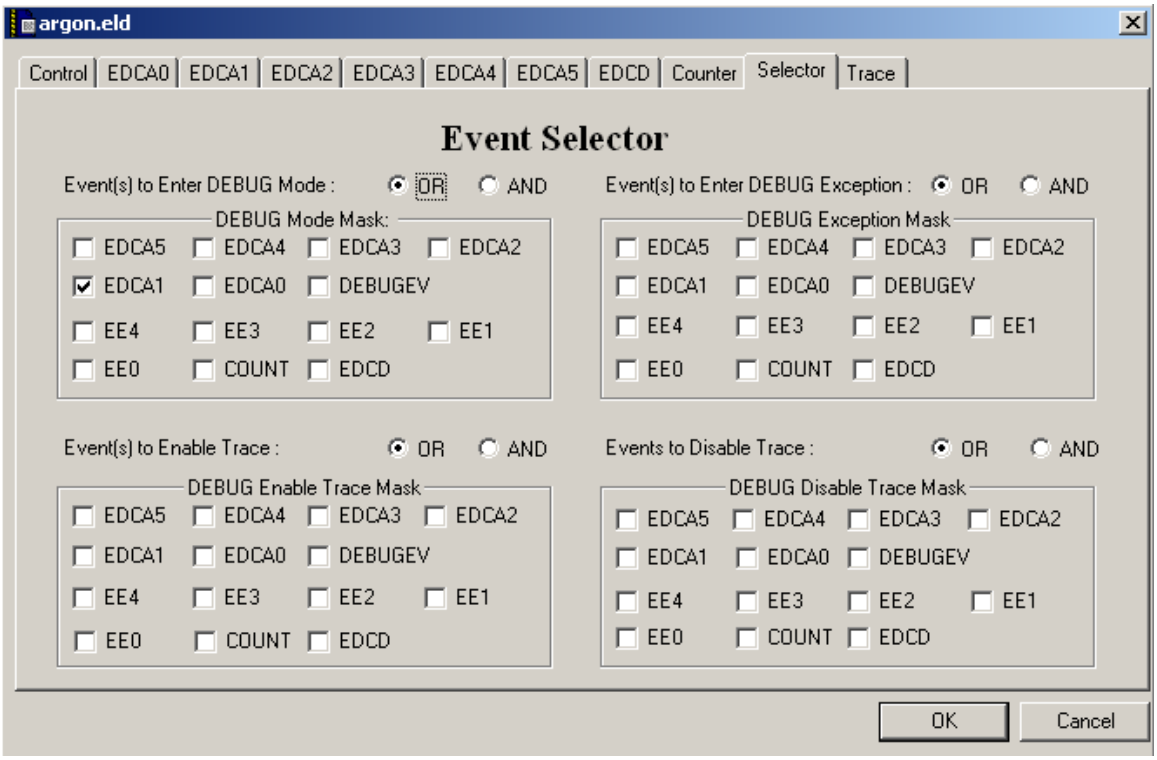

Figure 14: EOnCE Event Selector

3. Nexus Trace Setup
    a. Nexus Configuration: Figure 15: Nexus Configuration for Exception Trace shows the recommended configuration for Nexus Trace. This configures for the best possible interface to the RVT hardware and enables timestamps in the trace data.

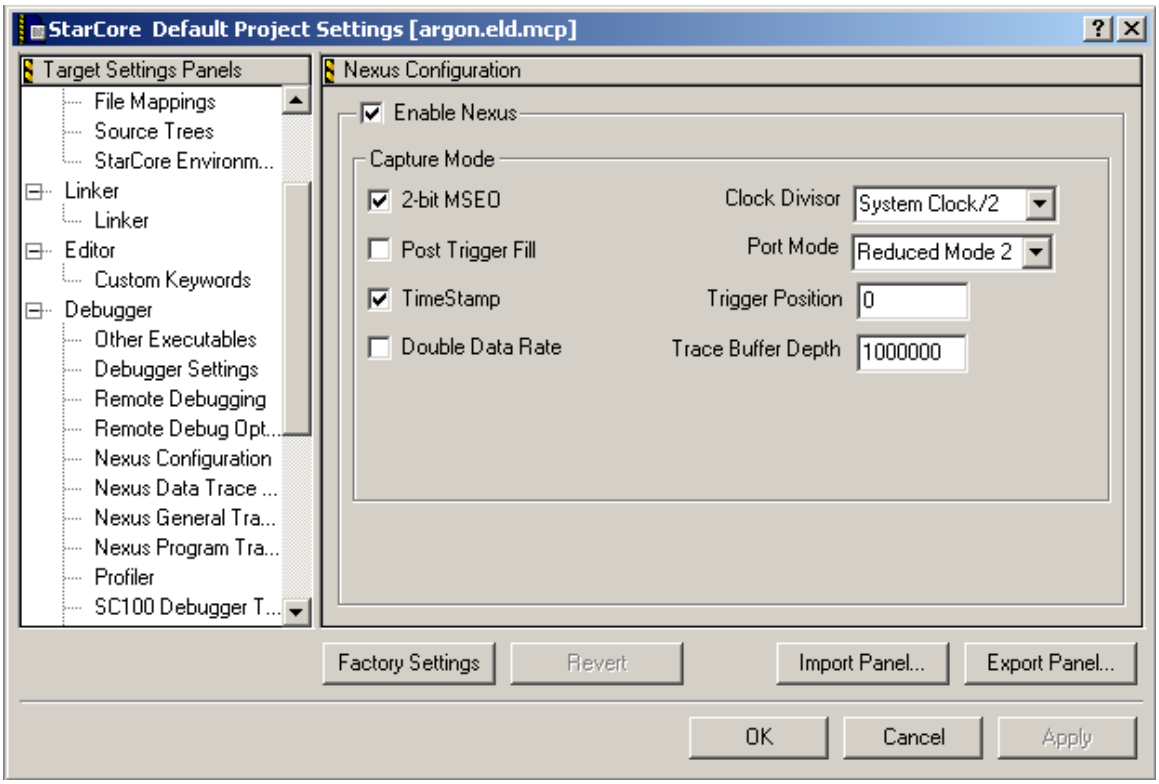

Figure 15: Nexus Configuration for Exception Trace

b. Nexus Program Trace Configuration: Figure 16: Nexus Program Trace for Exception Capture shows the setup for program trace. This setup dictates that tracing shall start when EDCA0 occurs and tracing shall stop when EDCA1 occurs. This is done to limit the program trace data collected.

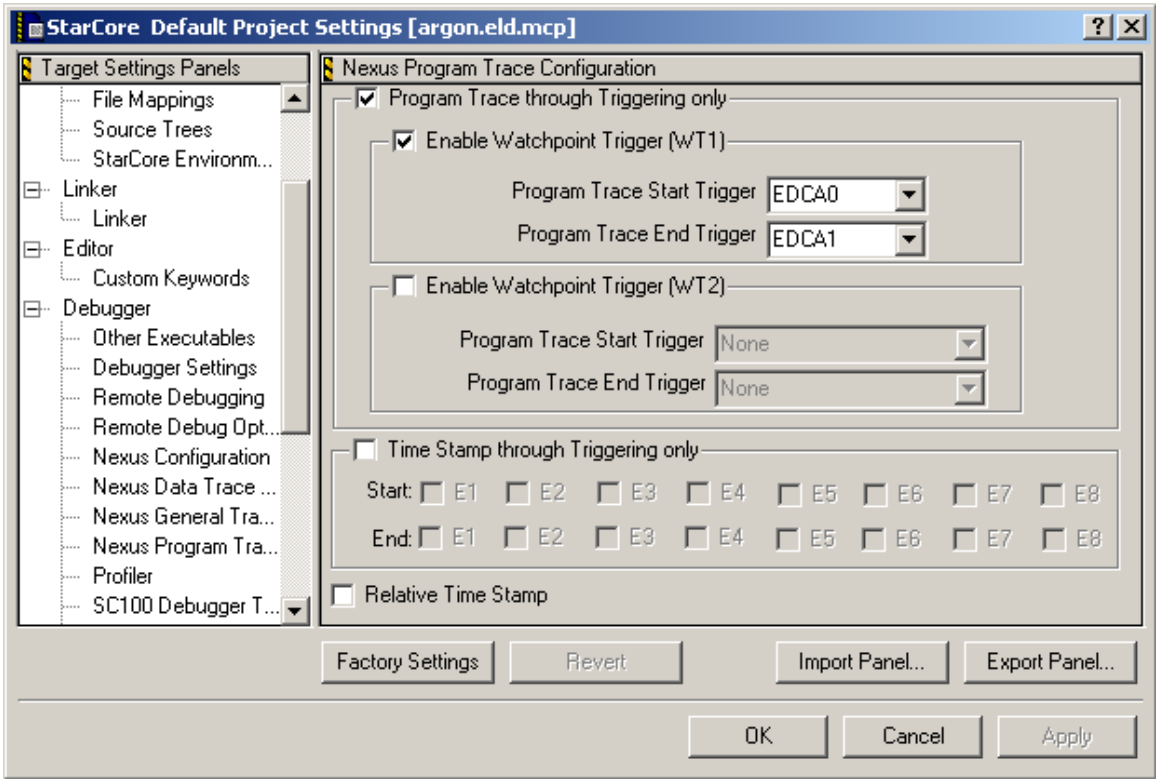

Figure 16: Nexus Program Trace for Exception Capture

c. Nexus Data Trace Configuration: Figure 17: Nexus Data Trace Configuration shows the Data Trace configuration. This essentially configures for no data tracing

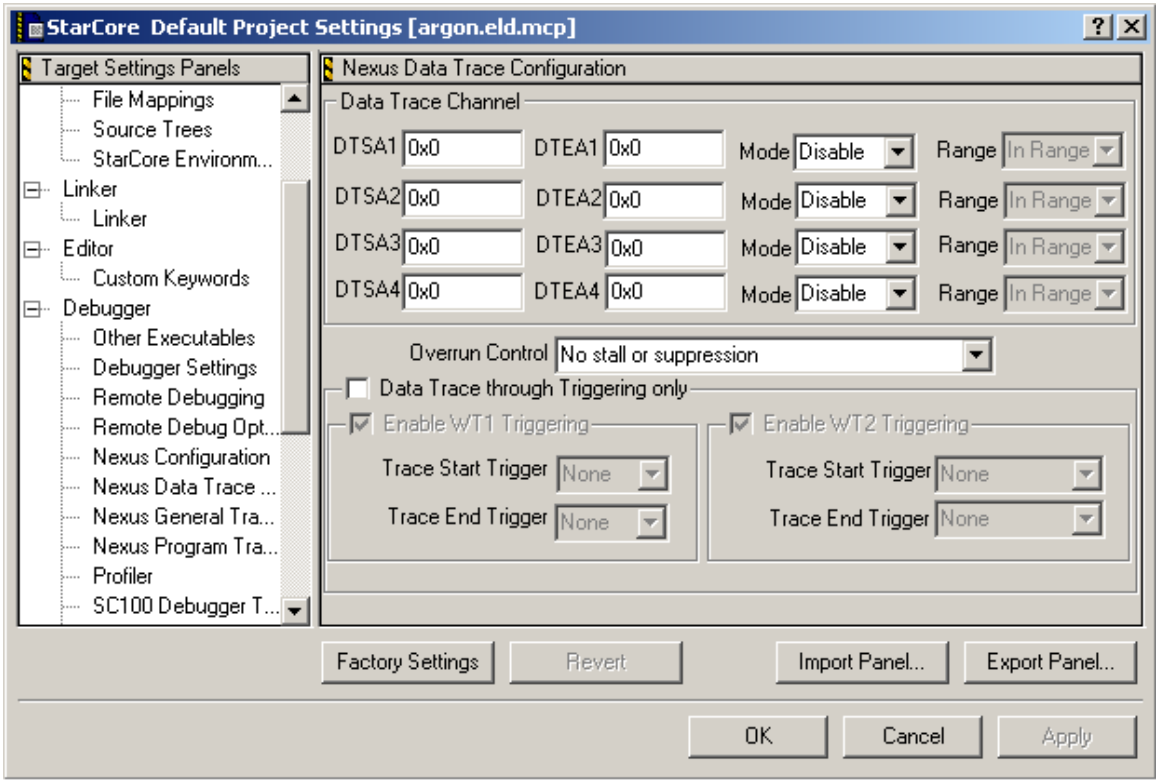

Figure 17: Nexus Data Trace Configuration

d. Nexus General Trace Configuration: Figure 18: Nexus General Configuration for Exception capture shows the settings for the General Trace configuration. These settings are similar to the Data Trace settings in that they are set to not generate additional trace messages.

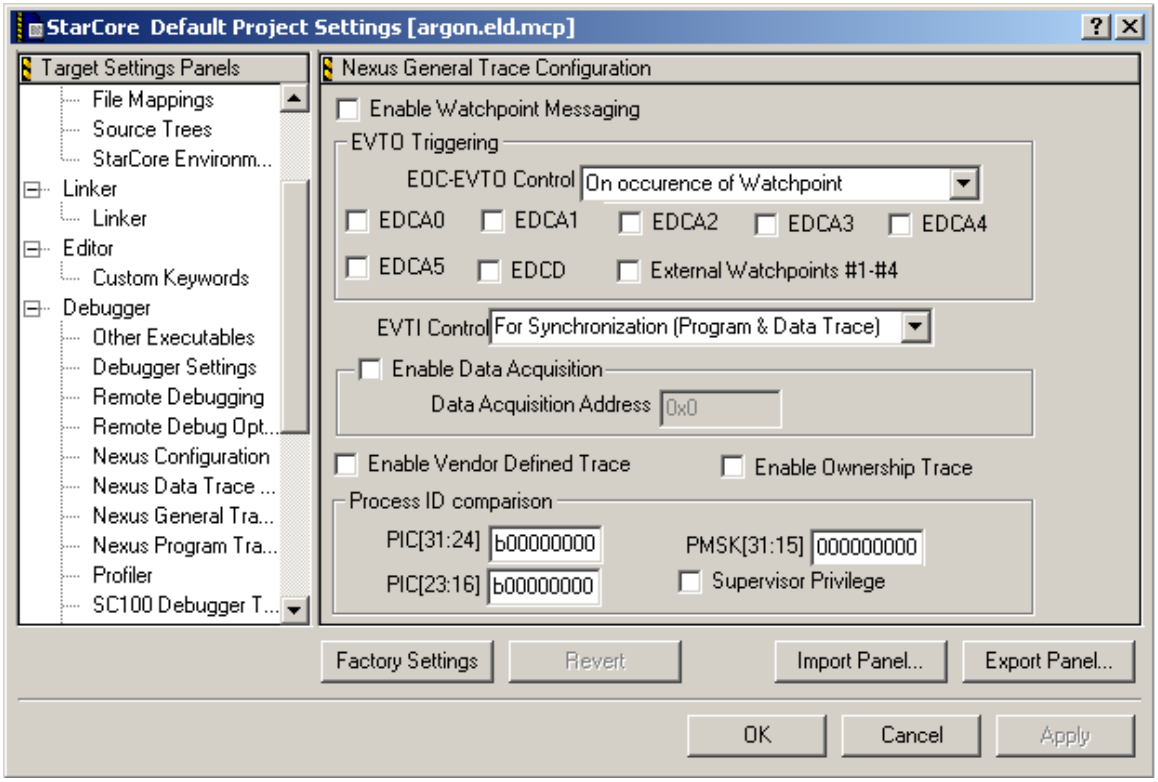

Figure 18: Nexus General Configuration for Exception capture

4. Execute Program
5. When Program hits exception it will halt.
6. View Trace Data in Program Mode

Data Trace: Monitor FIFO activity

In a multi-core embedded system such as Argon, data is often exchanged between the two cores by means of a DMA transfer to a memory buffer such as a FIFO. Data is also moved into the device via a hardware peripheral such as an A/D converter. One processor, an ISR or an RTOS task places data into a buffer, and another processor, ISR or RTOS task removes the data to be processed. A FIFO or other type of buffer is used to allow for variations in the processing time. It is often desirable to understand peak conditions for the FIFO or buffer use. In this case it is desired to see how frequently and deeply into a buffer data is being placed (written) before it is retrieved (read). Monitoring the entire buffer in terms of access type and setting watch points for the data buffer are used.

What is needed:

- CodeWarrior for StarCore/SDMA 1.0.3 or later
- ARM RVI (Real View ICE)+ARM RVT (Real View Trace)
- ARM RVT Ribbon to Mictor connector for ETM and Nexus (this is s special connector for use with a Nexus port)
- Argon + or Argon LV Hardware platform with JTAG port and 16 bit Nexus trace port on Mictor connector

- Software Project including
  - ELF debug file
  - Linker generated map file
  - source code (optional)

Steps to capture a data buffer trace:

1. Load the Project
   a. Identify the buffer or FIFO start address and length. Be sure to get the start and end address correct and adjust for the size of each member in the buffer. These addresses will be used later to setup the Nexus data trace.
2. Nexus Trace Setup
   a. Nexus Configuration setup is similar to previous example, this is simply setting the hardware interface between the Argon and the ARM RVT. Figure 15: Nexus Configuration for Exception Trace shows the correct setup for the Nexus interface.
   b. Nexus Data Trace Configuration is setup to trace a single data channel. It is configured to trace any read or write which occurs within a data buffer. The start and end addresses were found using the debugger in this example, but the map file also provides this information. Figure 19: Data Trace configuration shows the configuration for the data tracing.

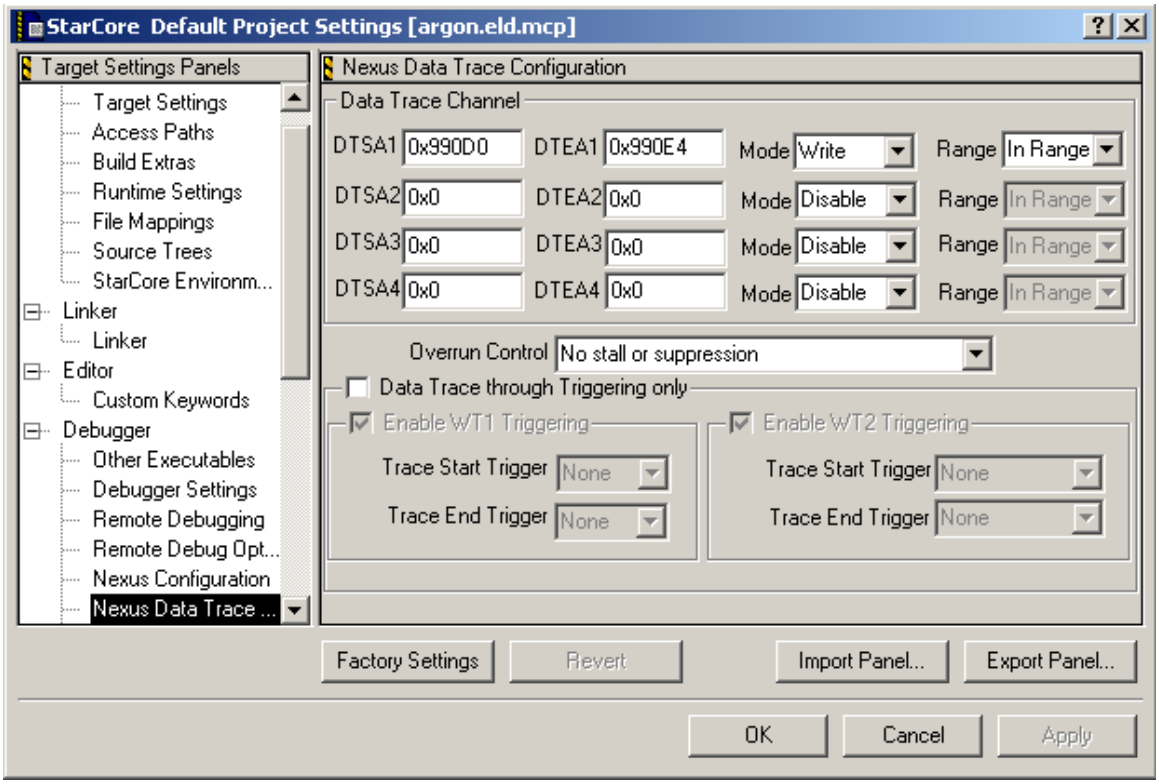

Figure 19: Data Trace configuration

   c. Program flow tracing is not used in this example so the setup for Program tracing should look like Figure 20: Program Flow Trace Setup.

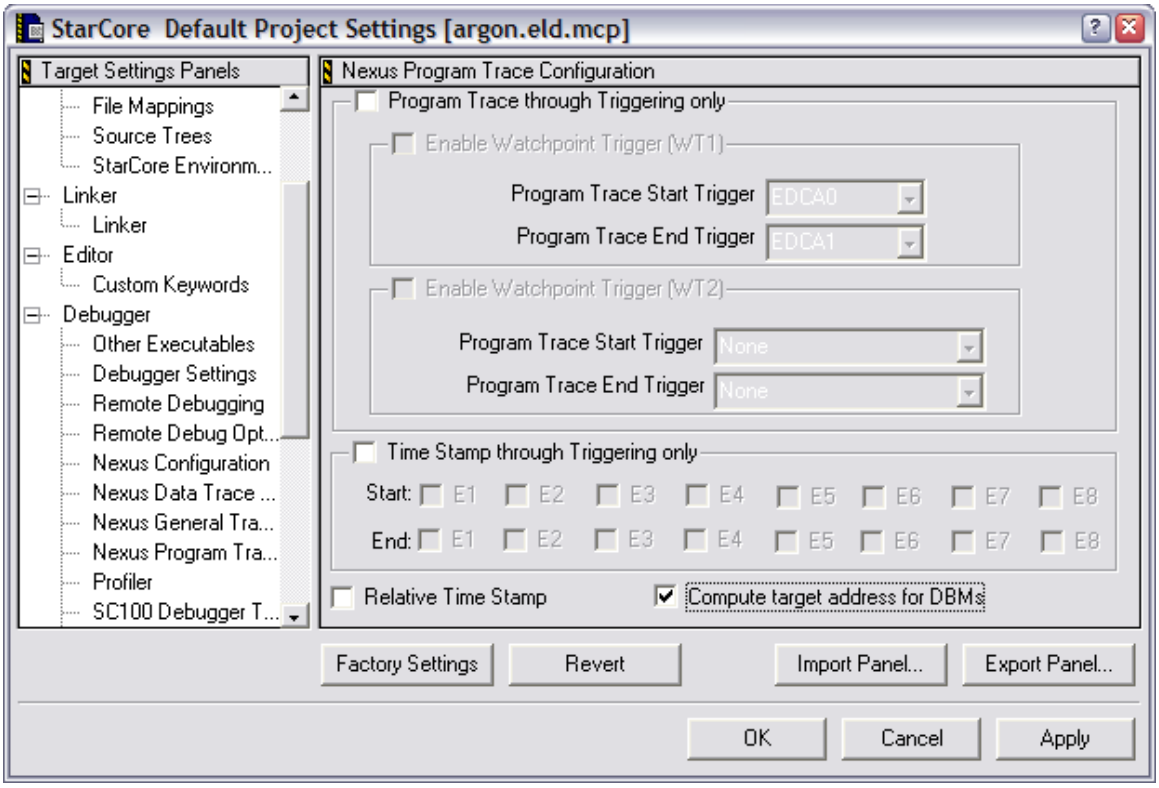

Figure 20: Program Flow Trace Setup

d. General Trace setup is also not used for this example. Figure 18: Nexus General Configuration for Exception capture shows the setup for General Trace.

3. Execute the Program
4. View the Trace Results in Data View mode

Program and Data Trace: Trace after Nth entry into a Function

In this case it is desired to trace program and data flow only after entry into a function for the Nth time. This example sets N = 10. The first 9 times a function is reached we don't care about what occurs, but on the 10th time the function is hit we would like to see program flow and some data tracing.

What is needed:

- CodeWarrior for StarCore/SDMA 1.0.3 or later
- ARM RVI (Real View ICE)+ARM RVT (Real View Trace)
- ARM RVT Ribbon to Mictor connector for ETM and Nexus (this is s special connector for use with a Nexus port)
- Argon + or Argon LV Hardware platform with JTAG port and 16 bit Nexus trace port on Mictor connector
- Software Project including

- ELF debug file
- Linker generated map file
- source code (optional)

Steps to capture an exception trace:

1. Load the project
    a. Configuration of EOnCE is performed real-time while connected to the hardware so a debug connection is necessary.
    b. Start the debug session, halting at the entry point will allow configuration of EOnCE to be used for this debug session without missing program execution.
    c. If there is a prior saved EOnCE configuration it will be used as the default configuration for EOnCE
2. Setup EOnCE events
    a. Setup the EOnCE Counter (see Figure 21: EOnCE Counter Setup)
        i. Count EDCA0 events
        ii.Enable the Counter
        iii.Set the Count value to 0x9

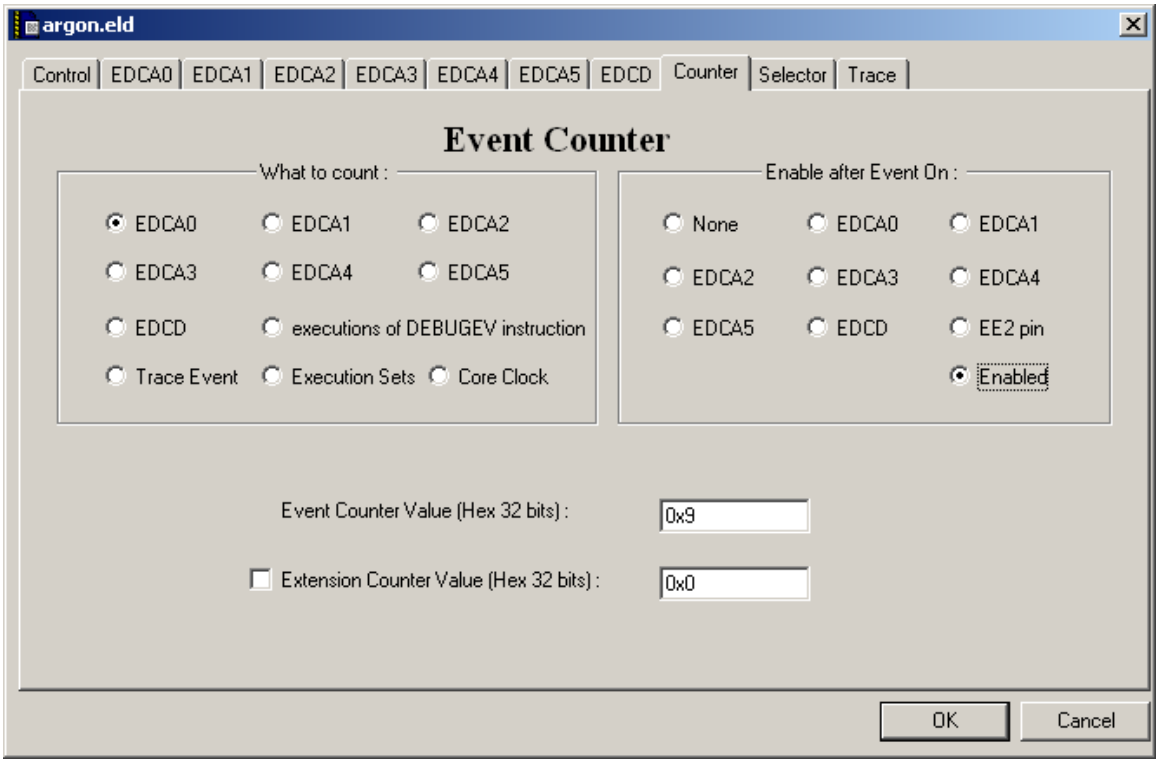

Figure 21: EOnCE Counter Setup

    b. Setup EOnCE Event EDCA0. This is the event we will be counted so it should be somewhere that is reached multiple times such as inside a loop, or a function which is called from many places (see Figure 12: Trigger Setup for End of Trace Collection)

i. Using the project's map or the debugger file search for the program address of a point in the source code inside the function of interest which will be counted before trace is initiated.
ii. In the EOnCE configuration window select the EDCA0 tab (see Figure 22: EDCA0 Setup)

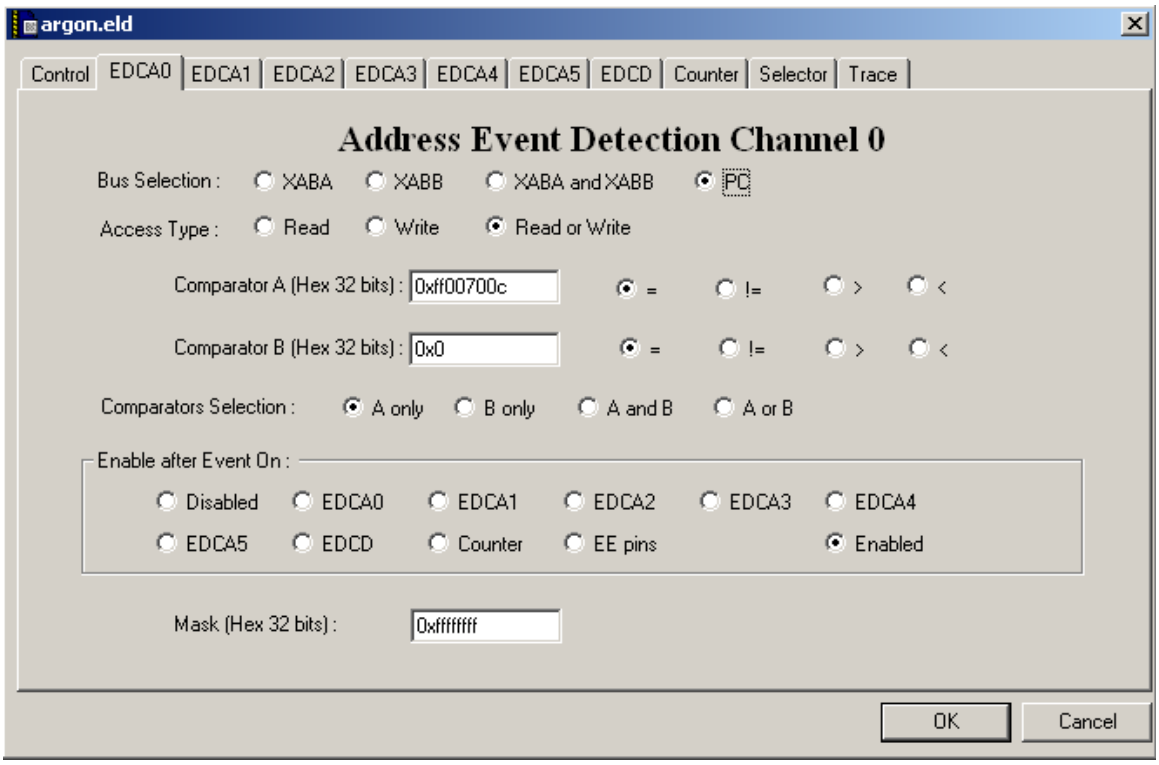

Figure 22: EDCA0 Setup

iii. Set Bus Selection to "PC"
iv. Set Access Type to "Read or Write"
v. Set Comparator A to Address found in previous step
vi. Set Comparators selection to "A only"
vii. Set Enable after Event On to "Enabled"

c. Setup EOnCE event EDCA1 to be used to trigger the start of tracing, but only after the counter has counted 9 times
   i. Find a point in the source code in near the start of the function which is being traced
   ii. In the EOnCE configuration window select the EDCA0 tab (see Figure 23: EDCA1 Setup)

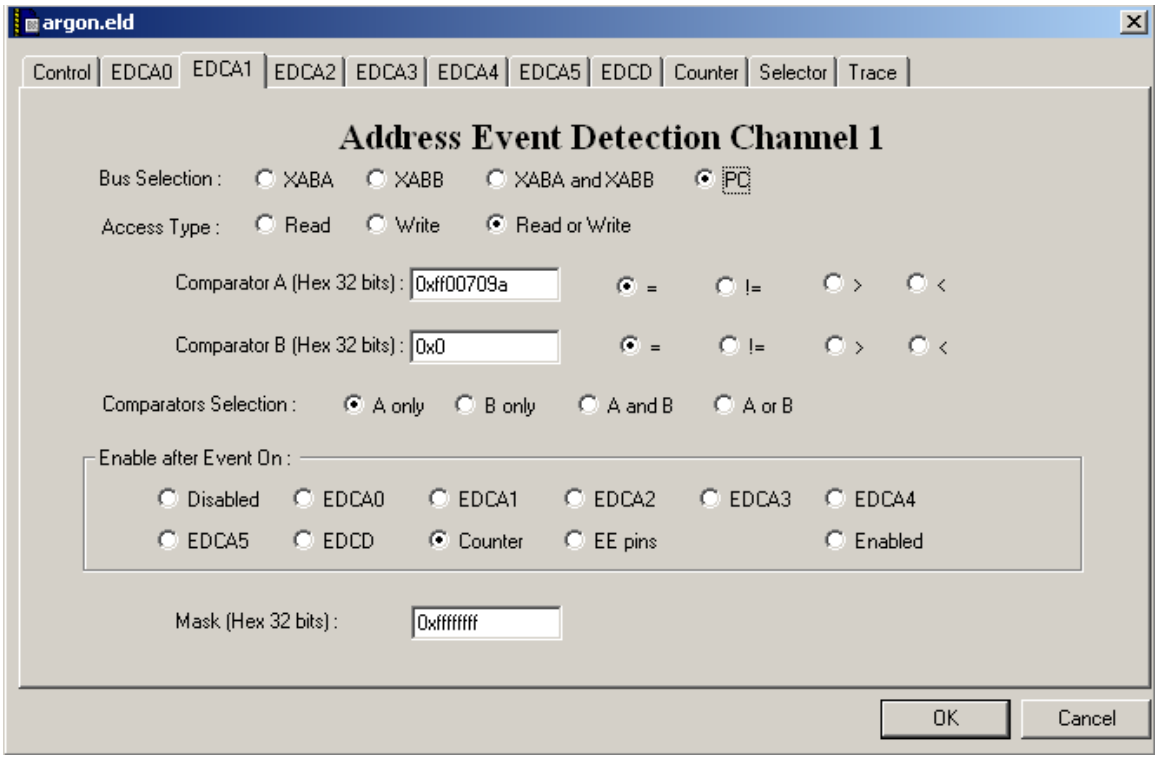

Figure 23: EDCA1 Setup

iii. Set Bus selection to “PC”
iv. Set Access Type to “Read or Write”
v. Set Comparator A to the address where trace should begin (note that the same address as EDCA0 can be used here also)
vi. Set Comparator Selection to “A only”
vii. Set Enable after Event On to “Counter”

d. Setup EOnCE event EDCA2 to be used to trigger the end of tracing
   i. In the EOnCE configuration menu select the EDCA2 tab (see Figure 24: EDCA2 Setup)

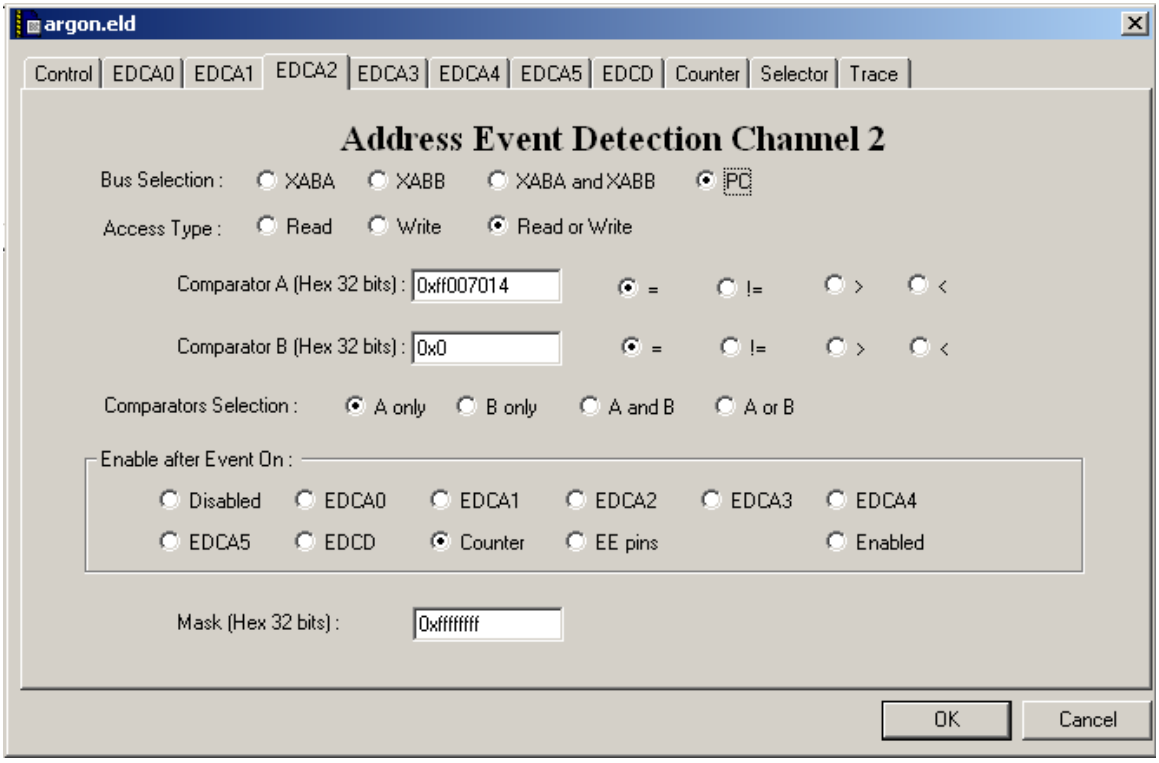

Figure 24: EDCA2 Setup

ii. Set Bus selection to “PC”
iii. Set Access Type to “Read or Write”
iv. Set Comparator A to the address where trace should end
v. Set Comparator Selection to “A only”
vi. Set Enable after Event On to “Counter”

e. Setup EOnCE to halt when EDCA2 occurs
   i. In the EOnCE configuration menu select the Selector Tab (see Figure 25: EOnCE Event Selector)
   ii. In the box labeled “Event(s) to Enter DEBUG Mode:”
      1. Set the mode to “OR”
      2. Select EDCA2
   iii. This will cause the StarCore processor to halt when the EDCA2 event occurs. This will also initiates download of the trace data.

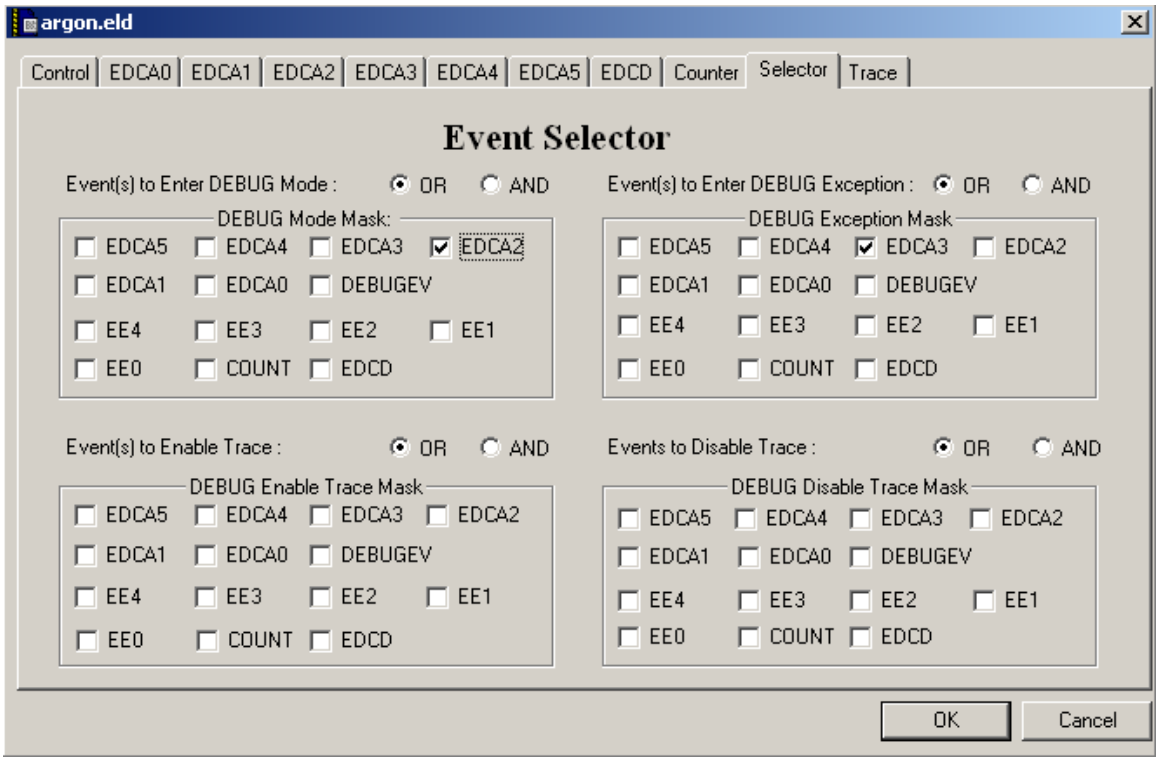

Figure 25: EOnCE Event Selector

3. Nexus Trace Setup
    a. Nexus Configuration: Figure 15: Nexus Configuration for Exception Trace shows the recommended configuration for Nexus Trace. This configures for the best possible interface to the RVT hardware and enables timestamps in the trace data.

    b. Nexus Program Trace Configuration: Figure 26: Nexus Program Trace Configuration shows the setup for program trace. This setup dictates that tracing shall start when EDCA1 occurs and tracing shall stop when EDCA2 occurs. This is done to limit the program trace data collected.

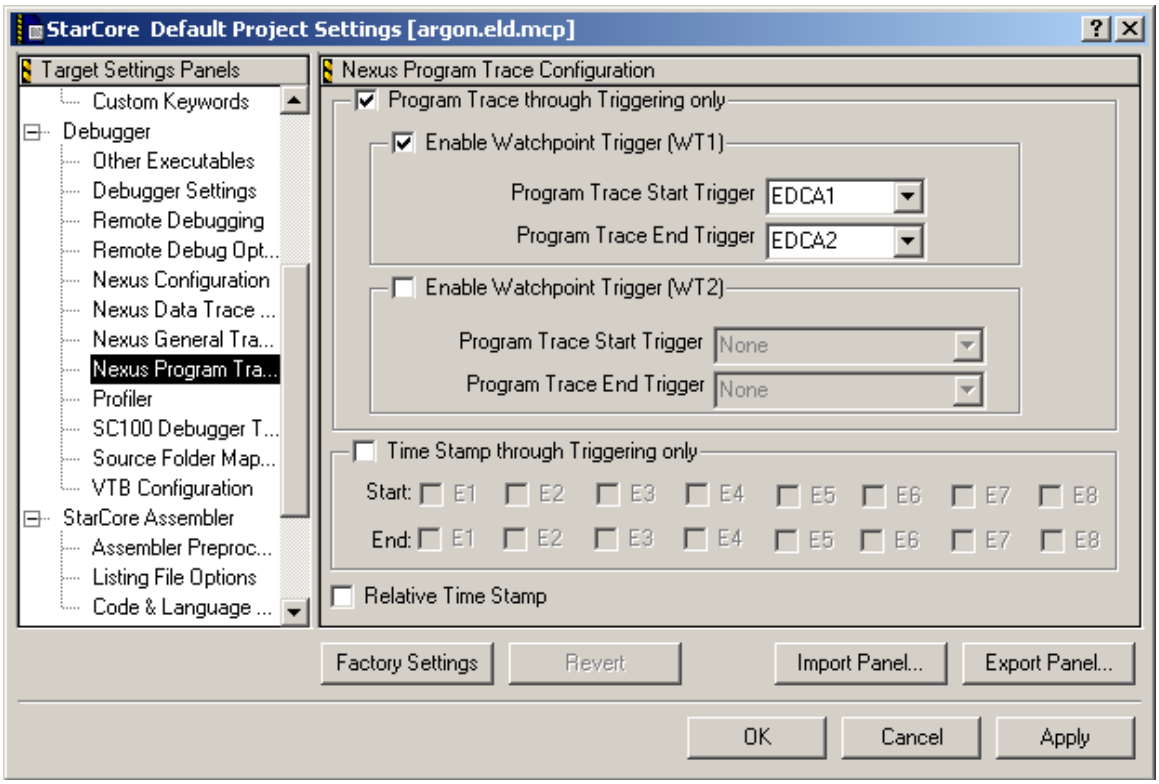

Figure 26: Nexus Program Trace Configuration

c. Nexus Data Trace Configuration: Figure 27: Nexus Data Trace Configuration shows the Data Trace configuration. This essentially configures a single channel of data tracing similar to the example in Section Data Trace: Monitor FIFO activity.

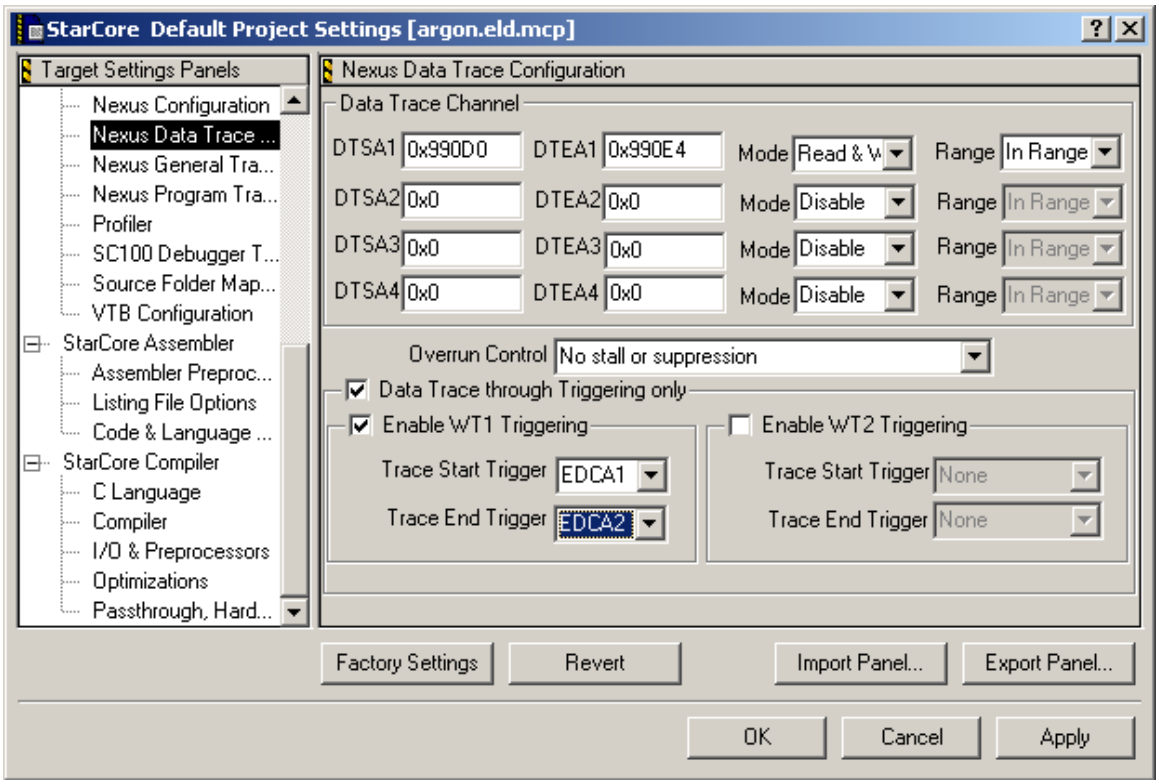

Figure 27: Nexus Data Trace Configuration

d. Nexus General Trace Configuration: Figure 18: Nexus General Configuration for Exception capture shows the settings for the General Trace configuration. These Nexus options are not used for this example.

4. Execute Program
5. When EDCA2 occurs, program will halt.
6. View Trace Data (example shown in Figure 28: Nexus Program Trace)

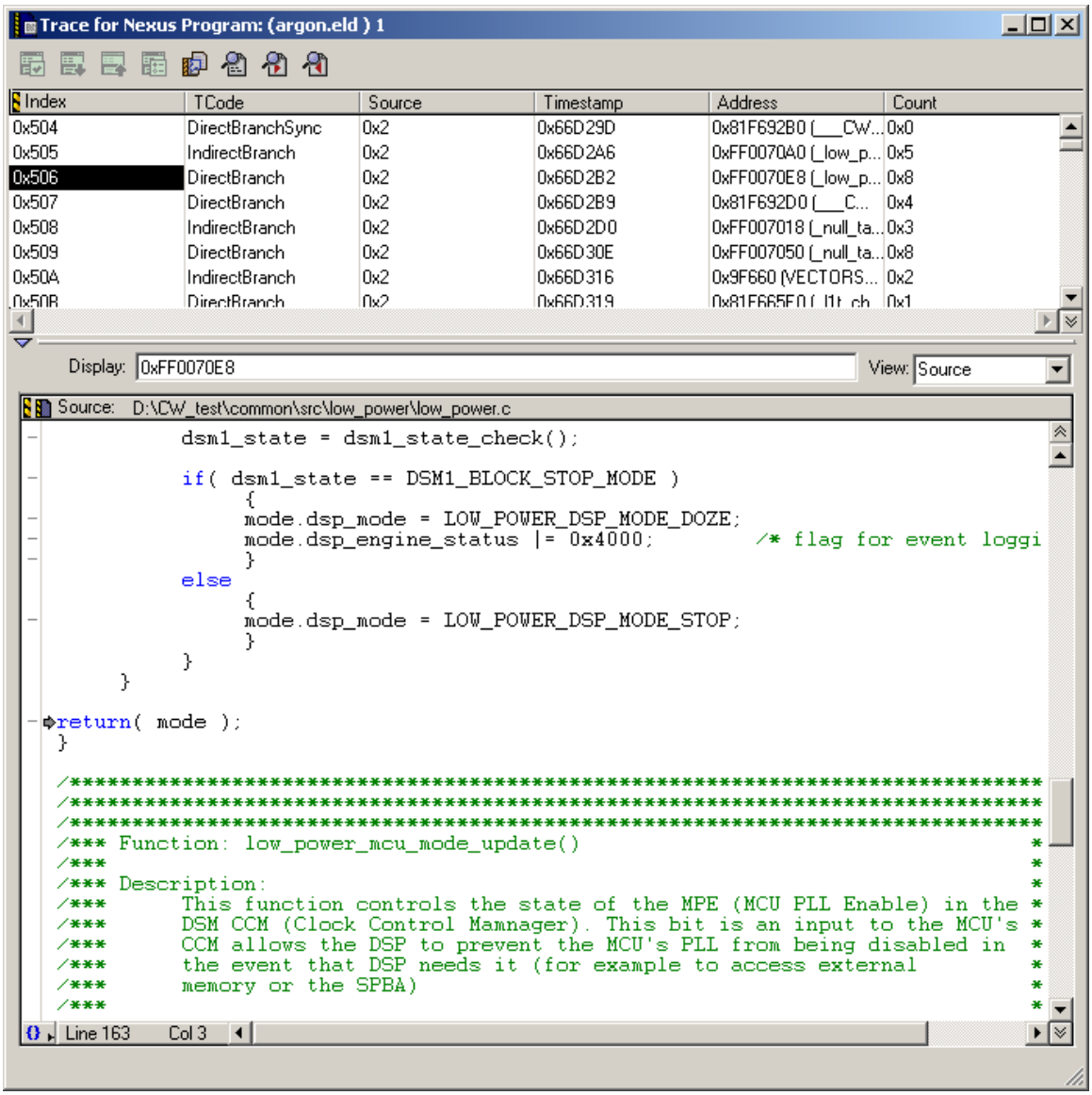

Figure 28: Nexus Program Trace

6 Troubleshooting Tips

Always check the hardware first! Be sure power is applied and connectors are seated correctly.

Problem: Not seeing any Trace data in the trace view!

- Verify Argon Device is a development device, and if it is not, verify that software can be run to configure the Nexus/SSP port as the primary nexus interface. Production Argon devices do not bring out the dedicated Nexus trace port, Development devices do.
- Inspect the Mictor connectors on the Nexus to RVT adapter PCB, verify no pins are bent as this can cause erroneous Trace data to occur or no trace data
- As always verify power is applied to the hardware being tested

- Try configuring a trigger to halt the StarCore, this will cause the program to halt if your trigger condition is met and will indicate if your trigger is being executed or is occurring.
- The ARM RVT is limited to a maximum clock rate of 125 MHz, if the DSP is configured to run faster than 250 MHz, then the Nexus Trace data could be getting clocked too fast for the RVT to handle.
  - Set the Nexus Clock divider to /4 instead of /2

Problem: Program is throwing EOnCE exceptions

This is a known issue and is caused by the presence of an ".eonce" file in the project directory. This file contains the last EOnCE configuration previously used, and will automatically load once the project is started. The exception is triggered as an EOnCE exception by the debugger.

- Workaround is to save the EOnCE configuration using the menu option: Debug->EOnCE->Save EOnCE Configuration, and delete or move the .eonce file from the project directory.
- Load the saved .cfg file after loading the project and initiating a debug session.

Problem: Trace data shows commands which do not make sense

This often occurs when the program on the target hardware does not match with the program in the loaded ELD file. Nexus decodes trace data based on the contents of the ELD file, when this does not match up, misalignments in the trace data can occur.

Problem: Nexus Events take a long time to download!

The Nexus Interface downloads serially over JTAG. For large amounts of trace data this can be time consuming. To mitigate this try:

- Limiting the amount of Trace collected by using triggered start and end points for trace collection
- Reduce the Trace Buffer Depth (See Figure 2: Nexus Configuration)

Problem: EOnCE configuration is demanding I turn on VTB

This occurs when the Virtual Trace Buffer Tracing mechanism in EOnCE is enabled. See Figure 11: EOnCE Trace Configuration and Table 10: EOnCE Trace Buffer Settings which describe this setting.

- Disabling this setting should prevent the tool from asking for VTB to be enabled and configured.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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

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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Order No.: AN3290
7/2006