Freescale Semiconductor
Application Note

Document Number: AN5123
Rev. 0, 05/2015

MPC5775K Optimizing Nexus Aurora SPT Trace for ADAS Applications

by: Curt Hillier

Contents

1 Introduction....1
2 Feature description....1
3 Data flow and Nexus trace diagnostic flow....2
4 Example: 6 SD ADC channel SDMA write trace....4
5 Nexus performance summary....8
6 Appendix A: Configuring Lauterbach TRACE32 debug tool and software....8

1 Introduction

Automotive electronic advancements in ADAS (Advanced Driver Assistance Systems) provide greater and greater reliability and safety, detecting dangerous conditions and preventing accidents and fatalities. ADAS solutions support a range of applications including Adaptive Cruise Control, Accident / Collision Avoidance, Lane Departure Warning, and Parking Assist. Freescale’s Automotive 77 GHz Radar transceiver chipset (MR2001) and MPC577xK Micro Controller Unit (MCU) provide a complete embedded radar system for automotive designs. These advanced solutions incorporate high-speed processing of reflected RADAR signals, enabling object detection with a high degree of precision.

Real time diagnostics for the RADAR stream is a key requirement for customers developing ADAS sensors based on the MR2001 and MPC5775K devices. Engineers need to capture what the RADAR sensors see in order to develop a high quality solution. This application note illustrates how Freescale's ICs along with the Lauterbach's TRACE32 ® debugger achieve this real time diagnostic requirement.

2 Feature description

To process the baseband analog signal provided by the RADAR devices (e.g. MR2001 Receiver), the MCP5775K first converts up to 8 channels of received analog data into digital 16-bit (12-bit data + 4 bits padding) results via 8 Sigma Delta Analog-to-Digital Converters (SD ADCs). The Sample Direct Memory Access (SDMA) block writes the SD ADC 16-bit results to device memory – either System RAM or e200z7 Tightly Couple Memory (TCM). In the diagram below, this is illustrated by a burst of 12 results sent from SD ADC via SDMA to memory. Once the results are written to memory, they are read out via Programmable DMA (PDMA) to the Signal Processing Toolkit (SPT) for Fast Fourier Transform (FFT) processing.

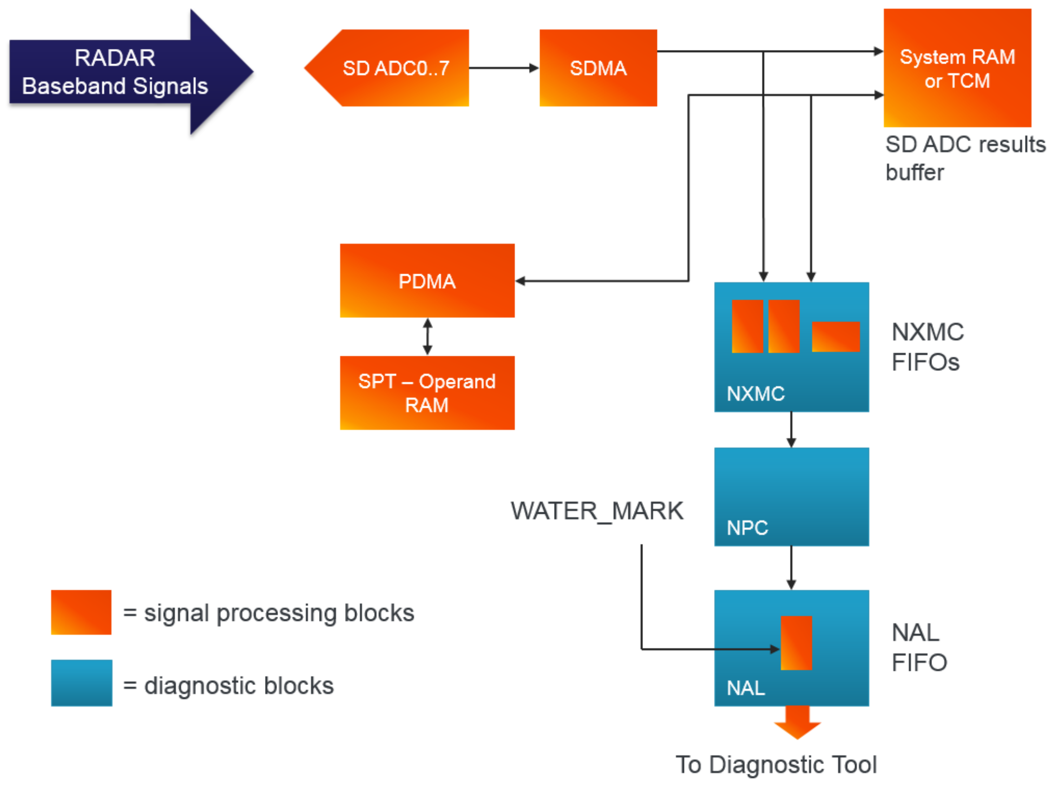


Figure 1. Nexus trace features for SPT SDMA

For diagnostics, the MPC5775K incorporates powerful Nexus DMA trace features to monitor the SDMA write transactions and the PDMA read and write transactions. In the above diagram, the NXMC receives a copy of the data written from SD ADC to Memory. The Nexus Crossbar Multi- Master Client (NXMC) packs each 64-bit data into a Nexus Data Write Message and sends the Data Write with Sync Message (DWSM) to the Nexus Port Controller (NPC) for transmission to the external debug tool via the Nexus Aurora Link (NAL) and Nexus Aurora Physical (NAP) blocks. The diagnostic tool buffers the data and displays the real-time captured RADAR results stream for the developer's analysis.

3 Data flow and Nexus trace diagnostic flow

The MPC5775K incorporates high performance analog-to-digital conversion, Direct Memory Access (DMA) data transfer, and hardware accelerated FFT signal processing. In the diagram below, these functions are performed by Sigma Delta ADC0..7, SDMA and PDMA blocks, and the Signal Processing Toolkit (SPT). These blocks make up the data flow portion of the solution.

In addition, the MPC5775K incorporates sophisticated Nexus Aurora trace capabilities. In the figure below, these functions are performed by the Nexus Crossbar Multi-Master Client (NXMC), the Nexus Port Controller (NPC), the Nexus Aurora Link (NAL) and the Nexus Aurora Phy (NAP).

To provide Nexus data trace messaging from bus masters on the master ports of the crossbar (XBAR) that do not inherently have a trace client, the NXMC monitors transactions flowing into the master port of the XBAR. Trace messages transmitted over the Nexus interface (or stored in trace memory) identify which NXMC generated the message as well as an identifier for the pre-concentrator source of the message. In the MPC5775K, NXMC0 supports LFAST, SIPI, and DMA data trace. NXMC1 supports FlexRay and Ethernet data trace. NXMC2 supports SD ADC and SPT data tracing via three pre-concentrators:

- SPT-ACQ: Traces SD ADC conversion data as it is transferred via Sample DMA (SDMA) to System RAM or TCM
- SPT-DMA: Traces Program DMA (PDMA) transfers between the SPT and System RAM or TCM
- SPT-SEQ: Traces Command Sequencer operations

The following block diagram illustrates the data flow and diagnostic flow portions of the digitized basedband radar signals.

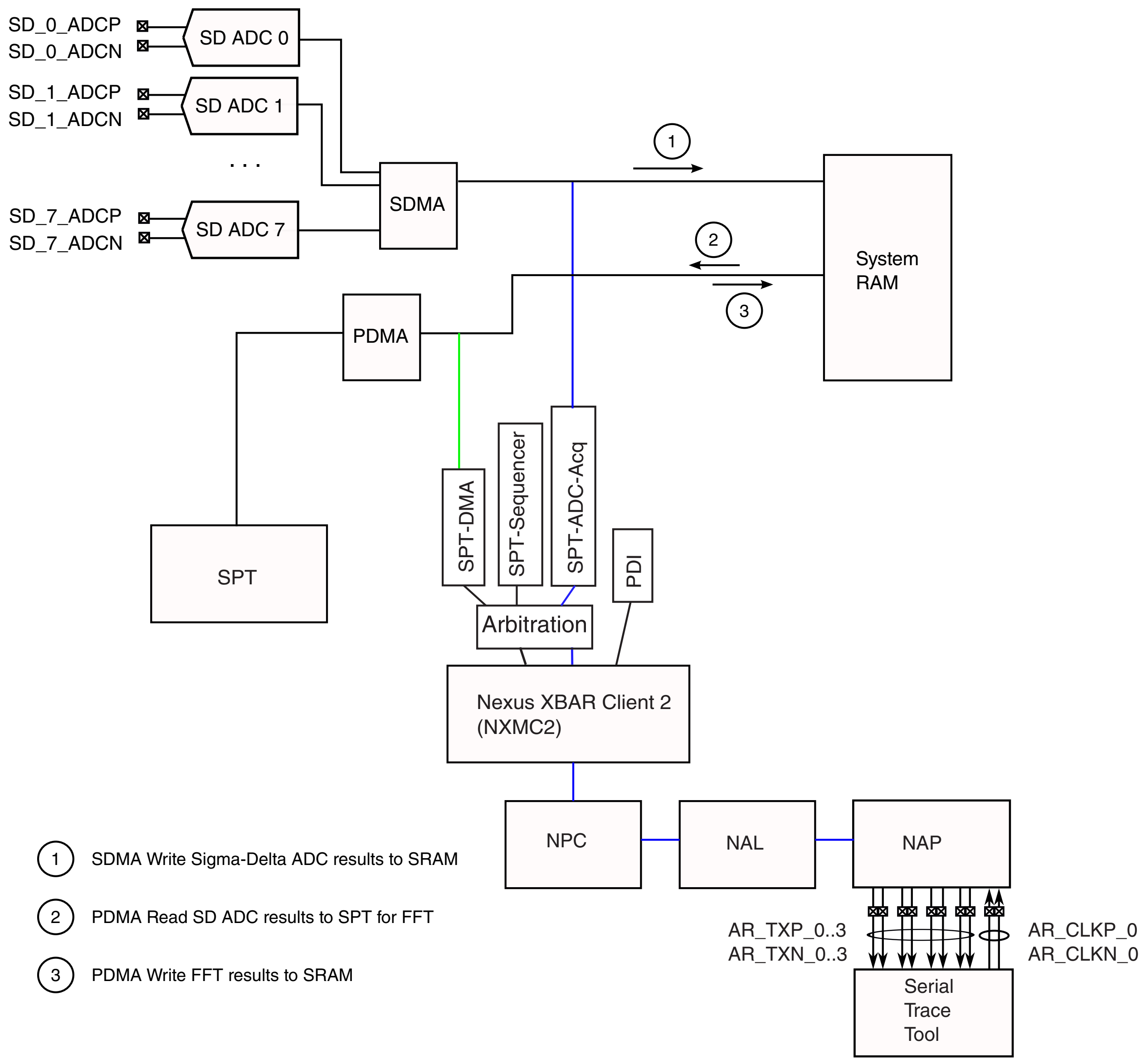


Figure 2. Data flow and diagnostic flow details

The MPC5775K incorporates a few controls to optimize Nexus trace flow. This application note provides recommended settings for the SPT DMA Control fields based on each example configuration. The optimized settings reduce the chance of Nexus FIFO overflows. These controls are configured in the MCB_NPC_SPECIAL_ENABLE.WATER_MARK field, the

MCB_MISC2.SPT_NEXUS_THROTTLE_CONTROL field, and the SPT_PDMA_CONTROL.PDMA_MAX_BURST_SIZE. These control fields apply to SDMA Write trace and PDMA Read/Write trace as shown in the following table:

Table 1. Register fields used to control Nexus FIFO watermark and PDMA transfer rate

Control Field	Range	SDMA Write	PDMA Read / Write
MCB_NPC_SPECIAL_ENABLE.WATER_MARK	0x0 to 0x7	yes	yes
MCB_MISC2.SPT_NEXUS_THROTTLE_CONTROL	0x0 to 0x3F	no	yes
SPT_PDMA_CONTROL.PDMA_MAX_BURST_SIZE	INCR4, INCR8, INCR16	no	yes

The MCB_NPC_SPECIAL_ENABLE.WATER_MARK tracks the NAL FIFO fill level. The NAL FIFO is 8 deep. When the FIFO fill level reaches the WATER_MARK setting, the Nexus Port Controller (NPC) stalls its 'grant' signal to the NXMC. When this occurs, incoming messages to the NAL will be held off, allowing the NAL to drain its FIFO before an overflow occurs.

NOTE

Both the NXMC2 and the NAL contain FIFOs to manage the SDMA and PDMA trace messages. If an overflow occurs, check the Nexus message Transfer Code (TCODE) and Source identifier (SRC). TCODE=08 is an error message. If the SRC = 0xE, then the overflow occurred in the NXMC2. If the SRC = 0xF, then the overflow occurred in the NAL.

For further reading, see the Reference Manual section: "Burst Size in SDMA for various Acquisition Modes/Channels Enabled in the DMA Arbiter". For the reader's convenience, the INCRn Burst Size detail is contained in the paragraph and table below.

In order to assess the impact of the number of SD ADC active channels on Nexus performance, the reader needs to first understand the relationship between number of SD ADC channels and DMA burst size. The table below shows the various burst sizes that are being used for SDMA. These are controlled by the different modes supported in SDMA. For burst sizes 4/8/16, the Arbiter initiates INCR4/INCR8/INCR16 respectively. The larger the burst size, the more likely the user can see an occassional overflow in the Nexus flow (NXMC FIFO overflow or NAL FIFO overflow).

Table 2. Transfer mode and number of channels effect on burst sizes

Modes	NUM_CHNL = 4	NUM_CHNL = 6	NUM_CHNL = 8
Interleaved	INCR16	INCR16	INCR16
Tile 4	INCR8	INCR12	INCR16
Tile 8	INCR8	INCR12	INCR16

4 Example: 6 SD ADC channel SDMA write trace

This example describes a 6 channel SD ADC use case. Data from the SD ADCs is collected into 64-bit data containers and then transferred via SDMA into either SRAM or the Tightly Coupled Memory (TCM) of the e200z7. In this use case, a total of 8 SD ADC conversions per channel are packed into two 64-bit data transfers in a TILE8 organization. TILE8 simply means 8 successive SD ADC results are copied into the SRAM or TCM results buffer on a per channel basis. The TILE8 alignment is shown below:

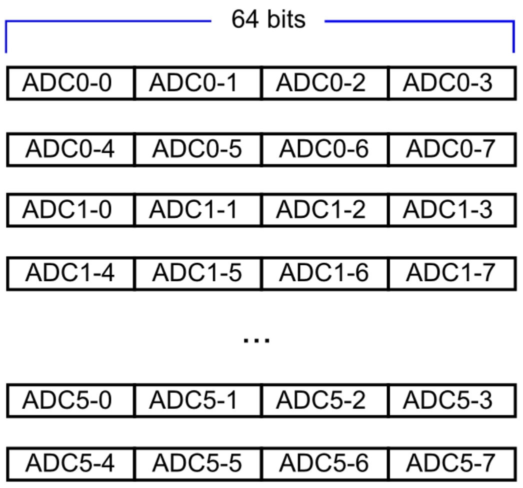


Figure 3. Alignment of SD ADC results using TILE8 configuration

For ADCn-m: 'n' is the SD ADC channel number and (m) is the SD ADC conversion number. This pattern repeats throughout the RADAR acquisition frame.

As the SDMA engine transfers SD ADC results to System RAM or in this case, TCM, the Nexus trace features can be configured to copy the DMA data to a diagnostic tool via Nexus Aurora. For customers using Lauterbach's TRACE32 trace tools, refer to Freescale's application note AN5005 for configuration details. As a reference, the following steps can be executed using the TRACE32 commands to configure SDMA trace:

Table 3. Lauterbach TRACE32 commands for configuring SDMA to TCM trace

TRACE32 Command	Description
NEXUS.BTM OFF	clear all other trace messages from e200 cores or other clients -- in this case, clear Branch Trace Message (BTM)
NEXUS.TimeStamps ON	set Nexus timestamping to ON. Timestamps are applied when a 64-bit data arrives at the NXMC2.
NEXUS.CLIENT3.MODE Write	set NXMC2 trace type to Write
NEXUS.CLIENT3.SELECT ALL	select all pre-concentractor IDs (SPT-ACQ, SPT-SEQ, and SPT-PDMA)
TrOnchip.Alpha TRACEDATACLIENT3	configure the trace on chip client id 'Alpha'
Break.Set 0x50800000--0x5080ffff /Write /Alpha	set the data range for the 'Alpha' data trace start address and data trace end address
Trace.List ALL	open the trace.list window - show decoded address, write SRC ID, data, and the Nexus message

The following C code is applicable to this use case:

main.c code:

```
#define NUM_CHIRPS 1        // number of chirps
#define NUM_SAMPLES 2560    // samples per chirp
void main( )
{
 //... pll init, mc mode init, cgm init, etc. function calls included here ...

 // configure Nexus FIFO watermark settings for Nexus trace
    MCB.NPC_SPECIAL_ENABLE.B.WATER_MARK = 0x2;      // set watermark to 2

// configure the HMASTER encoding
// use hmaster_enc_dis = 0 for this case
    if(hmaster_enc_dis == 1)
      SPT.SDMA_CTRL1.B.HMASTER_ENC_DIS = 1;              // set MID to 0xE.
    else
      SPT.SDMA_CTRL1.B.HMASTER_ENC_DIS = 0;             // set MID to SDADC ID

// enable 6 SD ADC channels, use TCM (0x50800000) for SD ADC results buffer
    SDADC_config_6ch(0x50800000, NUM_CHIRPS, NUM_SAMPLES);

// ... CTE function calls  ...

}
```

SDADC.c code:

```
/* -------------------------------------------------------------------------------
 FUNCTION    : SDADC_config_6ch
 INPUTS      : start_mem - location for SDMA to begin placing samples
               chirps     - number of chirps to acquire for complete frame
               samples    - number SDADC samples per chirp
 NOTES       : Configures the SDADCs for 6-channel conversion on ADC0..ADC5
               SDMA uses tile 4 organisation and place full frame in memory
---------------------------------------------------------------------------- */
void SDADC_config_6ch(uint32_t start_mem, uint32_t chirps, uint32_t samples){

    init_SRAM(start_mem, (samples*2*chirps));        // init the memory range

   // set FILT0CTRL for 10 Msps
     AFE_FILT0CTRL        = 0x00000000;
     AFE_FILT1CTRL        = 0x00000000;
     AFE_FILT2CTRL        = 0x00000000;
     AFE_FILT3CTRL        = 0x00000000;
     AFE_FILT4CTRL        = 0x00000000;
     AFE_FILT5CTRL        = 0x00000000;
     AFE_FILT6CTRL        = 0x00000000;
     AFE_FILT7CTRL        = 0x00000000;

   // 6 channels enabled
    AFE.ADCCTRL7.B.PWRDN = 0xC0;                                // powerup ADC0..5
    SPT.ACQ_GBL_CTRL_0.R = 0x05550000;                        // enable conversions on ADC0..5

    SPT.SDMA_CTRL0.R = start_mem;
    SPT.SDMA_CTRL1.B.SDMA_BUF_LEN = chirps;               // write chirps
    SPT.SDMA_CTRL1.B.DATA_ORG_CFG = 0x2;                  // Tile 8 sample organisation in
memory

    SPT.ACQ_GBL_CTRL_1.B.NUM_CHRP = chirps;              // no. of chirps in frame
    SPT.ACQ_GBL_CTRL_1.B.NUM_SMPL = (samples/8)-1;   // no. of samples in a chirp: (NUM+1)*8

}
```

Once the user has configured the trace tool for SDMA Write tracing of the SD ADC results buffer and executes the corresponding C code, the SD ADCs are enabled and will convert the incoming analog baseband RADAR signals into digital results. As the results are transferred via SDMA to TCM, the resulting trace information can be obtained. See the table below for an example of the 6 channel trace use case results:

Table 4. Data trace results for the 6 channel use case

Address	Nexus Message
0x50800000	TCODE=3C SRC=E NXMC DWSM MID=0 DSZ=4 F-ADDR=50800000 DATA=98FFA0FFF8FF0000
0x50800008	TCODE=3A SRC=E NXMC DWM MID=0 DSZ=4 U-ADDR=00000008 DATA=98FF90FF98FFA0FF
0x50800010	TCODE=3A SRC=E NXMC DWM MID=1 DSZ=4 U-ADDR=00000018 DATA=5000500010000000
0x50800018	TCODE=3A SRC=E NXMC DWM MID=1 DSZ=4 U-ADDR=00000008 DATA=5800500058004800
0x50800020	TCODE=3A SRC=E NXMC DWM MID=2 DSZ=4 U-ADDR=00000038 DATA=A0FF90FFF0FF0000

Table continues on the next page...

Table 4. Data trace results for the 6 channel use case (continued)

0x50800028	TCODE=3A SRC=E NXMC DWM MID=2 DSZ=4 U-ADDR=00000008 DATA=98FF98FF98FFA0FF
0x50800030	TCODE=3A SRC=E NXMC DWM MID=3 DSZ=4 U-ADDR=00000018 DATA=0800080000000000
0x50800038	TCODE=3A SRC=E NXMC DWM MID=3 DSZ=4 U-ADDR=00000008 DATA=1800100010001000
0x50800040	TCODE=3A SRC=E NXMC DWM MID=8 DSZ=4 U-ADDR=00000078 DATA=7800780008000000
0x50800048	TCODE=3A SRC=E NXMC DWM MID=8 DSZ=4 U-ADDR=00000008 DATA=7800780080007800
0x50800050	TCODE=3A SRC=E NXMC DWM MID=9 DSZ=4 U-ADDR=00000018 DATA=0000000000000000
0x50800058	TCODE=3A SRC=E NXMC DWM MID=9 DSZ=4 U-ADDR=00000008 DATA=0000000008001000
0x50800060	TCODE=3A SRC=E NXMC DWM MID=0_DSZ=4 U-ADDR=00000038 DATA=98FFA0FFA8FFA0FF
0x50800068	TCODE=3A SRC=E NXMC DWM MID=0 DSZ=4 U-ADDR=00000008 DATA=A0FFA0FF98FF98FF
0x50800070	TCODE=3A SRC=E NXMC DWM MID=1 DSZ=4 U-ADDR=00000018 DATA=5800600050005000

Where the Nexus Message contents are:

- TCODE=3C is a Data Write with Sync message (F-ADDR is the full address)
- TCODE=3A is a Data Write message (U-ADDR is the relative address)
- MID=n is the SD ADC instance number
- DSZ=4 is the data size (64 bits)
- DATA is the SD ADC data results

In the above results, it can be difficult to see the stream of results for a single SD ADC. The TRACE32 software supports a powerful scripting language, PRACTICE, which can extract a single SD ADCs data results. The instructions shown below extract the address and data for SD ADC 1:

```
Trace.Find CYcle wr-m01    // find first write cycle for wr-m01
RePeat 0
(
  if !FOUND()
    ENDDO
  &record=TRACK.RECORD()                           // get record number
  &addr_result=Analyzer.RECORD.ADDRESS(&record)    // get data
  &data_result=Analyzer.RECORD.DATA(&record)       // get data
  PRINT "ADDR: " &addr_result " " "DATA: " FORMAT.HEX(16.,&data_result)
  Trace.Find                                       // find next write cycle
)
```

The results for this PRACTICE script are shown in the following table:

Table 5. PRACTICE script results

ID	SD ADC 1 raw results
1	ADDR: D:0x50800010 DATA: 0050005800500030
2	ADDR: D:0x50800018 DATA: 0040003000500058
3	ADDR: D:0x50800070 DATA: 0050005000600058
4	ADDR: D:0x50800078 DATA: 0050004800280040
5	ADDR: D:0x508000D0 DATA: 0038004000580048
6	ADDR: D:0x508000D8 DATA: 0048005000500048
7	ADDR: D:0x50800130 DATA: 0050005800580030
8	ADDR: D:0x50800138 DATA: 0040003800400058
9	ADDR: D:0x50800190 DATA: 0048004000500050
10	ADDR: D:0x50800198 DATA: 0058006000500058

Table continues on the next page...

Table 5. PRACTICE script results (continued)

11	ADDR: D:0x508001F0 DATA: 0040005800380028
12	ADDR: D:0x508001F8 DATA: 0038003000500040
13	ADDR: D:0x50800250 DATA: 0058005000480050
14	ADDR: D:0x50800258 DATA: 0050005000580050
15	ADDR: D:0x508002B0 DATA: 0030003800400050
16	ADDR: D:0x508002B8 DATA: 0058005000500050

5 Nexus performance summary

The following table shows various use cases depending on the number of SD ADC channels enabled and the memory used for the SD ADC results. In some cases, for smaller channel counts, the Nexus blocks will not experience overflows. For larger number of channels (6 or 8) the burst size is large enough the Nexus blocks will experience occasional overflows. In these cases, using TCM instead of SRAM will reduce the overflow rate significantly.

Table 6. Optimized settings to minimize Nexus overflows

Number of SD ADC channels	Burst Size	Memory used	WATER_MARK setting	BWE_SPT	Overflow observed?	Approximate overflow rate (out of 1 million transfers)
2	4	TCM or SRAM	2	Don’t care	No	0
4	8	TCM or SRAM	2	Don’t care	No	0
6	12	SRAM	2	Don’t care	Yes (10 NXMC, 0 NAL)	10
6	12	TCM	2	Don’t care	Yes	<1
8	16	SRAM	4	0 (disabled)	Yes (out of 20 overflows, 12 NXMC, 8 NAL)	200
8	16	SRAM	2	0 (disabled)	Yes (out of 20 overflows, 20 NXMC, 0 NAL)	16,000
8	16	TCM	2	0 (disabled)	Yes (1 NAL)	<1

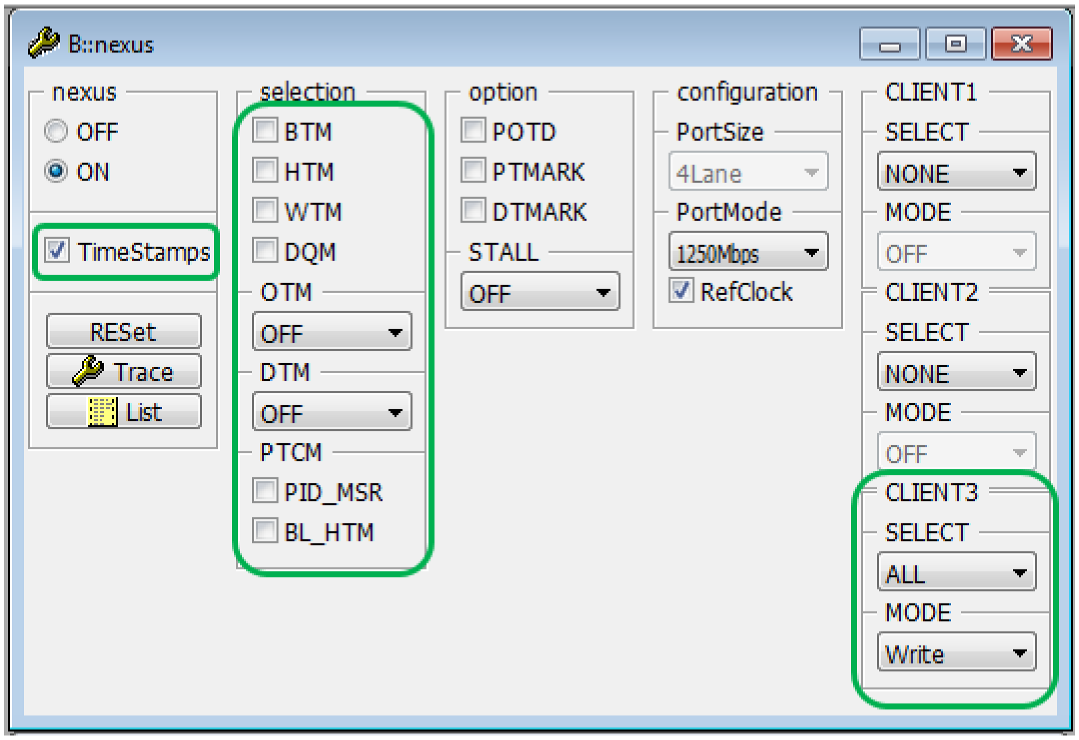
where:

WATER_MARK = MCB_NPC_SPECIAL_ENABLE[WATER_MARK]
BWE_SPT = PCM_IAHB_BE5[BWE_SPT]

6 Appendix A: Configuring Lauterbach TRACE32 debug tool and software

The user can configure Lauterbach's TRACE32 using the following steps.

Step 1: Configure the Nexus window. Select Trace -> Nexus Setttings...



B::nexus

nexus: OFF [✓ ON]

[✓] TimeStamps

RESet | Trace | List

selection: [ ] BTM, [ ] HTM, [ ] WTM, [ ] DQM

OTM: OFF

DTM: OFF

PTCM: [ ] PID_MSR, [ ] BL_HTM

option: [ ] POTD, [ ] PTMARK, [ ] DTMARK

STALL: OFF

configuration: PortSize: 4Lane; PortMode: 1250Mbps; [✓] RefClock

CLIENT1: SELECT: NONE; MODE: OFF

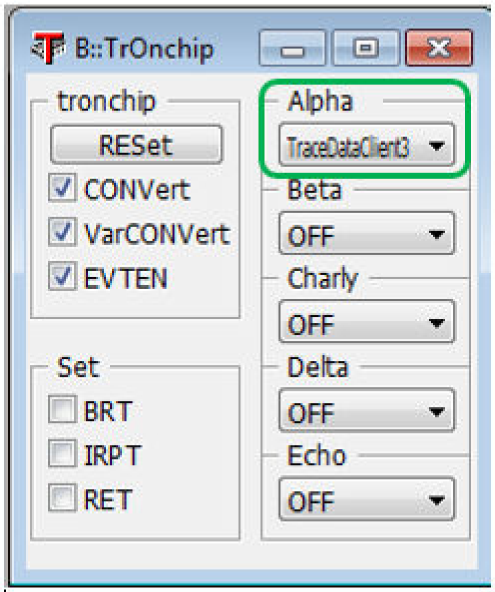
CLIENT2: SELECT: NONE; MODE: OFF

CLIENT3: SELECT: ALL; MODE: Write

Figure 4. Configure Nexus settings

In step 1, we are selecting TimeStamps, clearing all 'selection' settings including BTM, HTM, WTM, and DQM, and we configure CLIENT3 as shown in the figure.

Step 2. Configure the TrOnChip settings.



B::TrOnchip

tronchip: RESet; [✓] CONVert; [✓] VarCONVert; [✓] EVTEN

Set: [ ] BRT, [ ] IRPT, [ ] RET

Alpha: TraceDataClient3

Beta: OFF

Charly: OFF

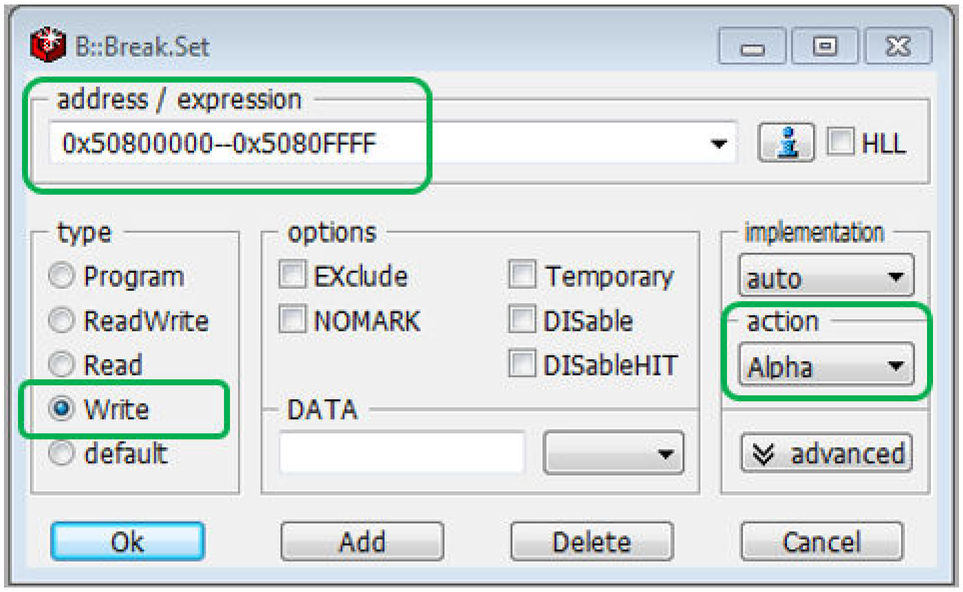
Delta: OFF

Echo: OFF

Figure 5. Configure TrOnchip settings

In Step 2, select TraceDataClient3 for Alpha. We will later tie this to the BREAK.SET configuration step.

Step 3. Configure the data window to trace via Break -> Set...



address / expression: 0x50800000--0x5080FFFF

type: Program, ReadWrite, Read, [✓ Write], default

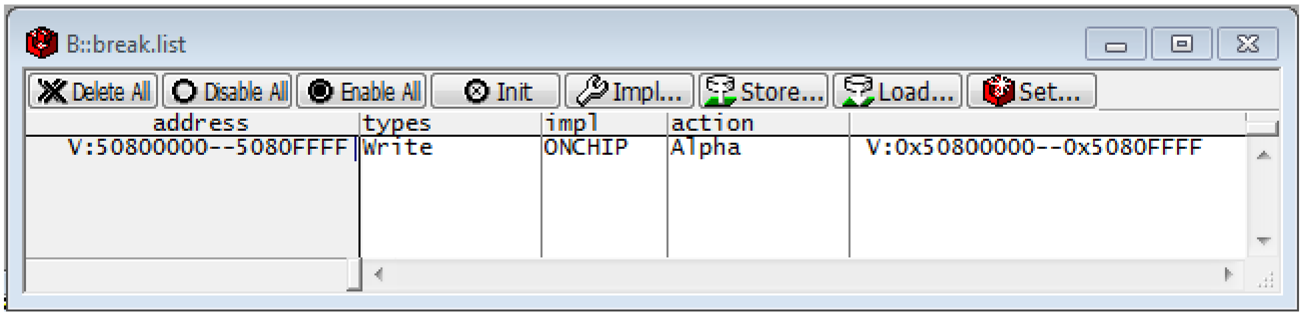
implementation: auto

action: Alpha

Figure 6. Configure the memory range to trace

In Step 3, we configure the memory range to trace by programming the range into the address / experssion box. Then, select Write for the type and Alpha for the action. Finally, click OK. This will set the DTSAn (Data Trace Start Address) and DTEAn (Data Trace End Address) in the Nexus Crossbar Multi-master Client (NXMC_2). It also sets the NXMC_2 DTC.RWT1 register field to write.

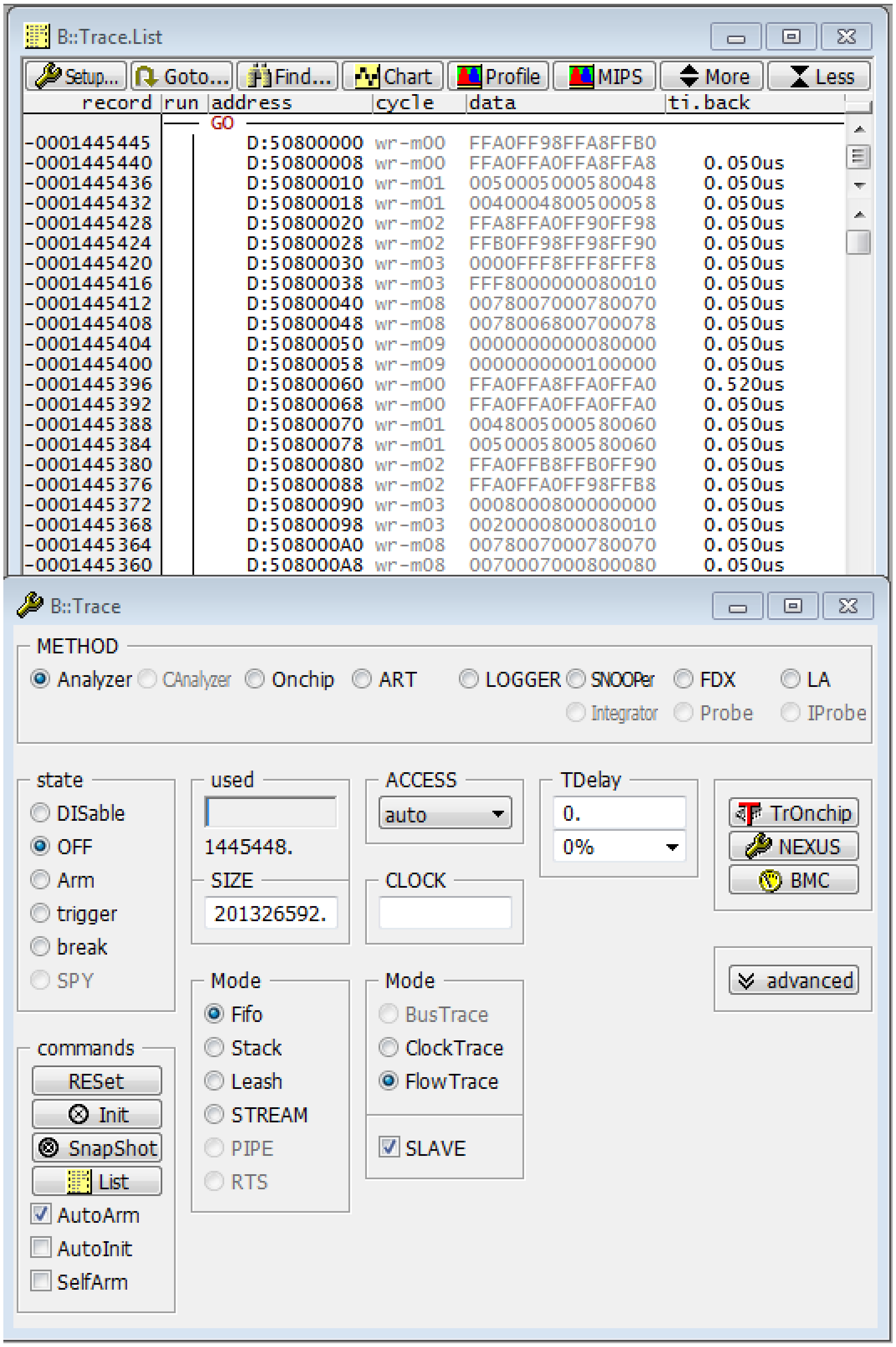
Step 4. Confirm the data range selections via the Break -> List... window.



```
      address          types      impl      action
  V:50800000--5080FFFF Write      ONCHIP    Alpha       V:0x50800000--0x5080FFFF
```

Figure 7. Confirm settings in Break -> List window

Step 5. Execute code and trace the SDMA writes to TCM



```
      record |run |address        |cycle    |data              |ti.back
              ─── GO
-0001445445        D:50800000 wr-m00 FFA0FF98FFA8FFB0
-0001445440        D:50800008 wr-m00 FFA0FFA0FFA8FFA8    0.050us
-0001445436        D:50800010 wr-m01 0050005000580048    0.050us
-0001445432        D:50800018 wr-m01 0040004800500058    0.050us
-0001445428        D:50800020 wr-m02 FFA8FFA0FF90FF98    0.050us
-0001445424        D:50800028 wr-m02 FFB0FF98FF98FF90    0.050us
-0001445420        D:50800030 wr-m03 0000FFF8FFF8FFF8    0.050us
-0001445416        D:50800038 wr-m03 FFF8000000080010    0.050us
-0001445412        D:50800040 wr-m08 0078007000780070    0.050us
-0001445408        D:50800048 wr-m08 0078006800700078    0.050us
-0001445404        D:50800050 wr-m09 0000000000080000    0.050us
-0001445400        D:50800058 wr-m09 0000000000100000    0.050us
-0001445396        D:50800060 wr-m00 FFA0FFA8FFA0FFA0    0.520us
-0001445392        D:50800068 wr-m00 FFA0FFA0FFA0FFA0    0.050us
-0001445388        D:50800070 wr-m01 0048005000580060    0.050us
-0001445384        D:50800078 wr-m01 0050005800580060    0.050us
-0001445380        D:50800080 wr-m02 FFA0FFB8FFB0FF90    0.050us
-0001445376        D:50800088 wr-m02 FFA0FFA0FF98FFB8    0.050us
-0001445372        D:50800090 wr-m03 0008000800000000    0.050us
-0001445368        D:50800098 wr-m03 0020000800080010    0.050us
-0001445364        D:508000A0 wr-m08 0078007000780070    0.050us
-0001445360        D:508000A8 wr-m08 0070007000800080    0.050us
```

METHOD: [✓ Analyzer] CAnalyzer Onchip ART LOGGER SNOOPer FDX LA Integrator Probe IProbe

state: DISable [✓ OFF] Arm trigger break SPY

used: 1445448.

SIZE: 201326592.

ACCESS: auto

CLOCK:

TDelay: 0. / 0%

commands:

[✓ AutoArm] AutoInit SelfArm

Mode: [✓ Fifo] Stack Leash STREAM PIPE RTS

Mode: BusTrace ClockTrace [✓ FlowTrace]

[✓ SLAVE]

Figure 8. Execute code and trace data

In Step 5, two windows are shown - the Trace -> List window and the Trace window. The Trace -> List window contains the Address, Cycle (master ID), and the resulting data. In this case, the master IDs wr-m00, wr-m01, wr-m02, wr-m03, wr-m08, and wr-m09 correspond to Sigma Delta ADC instances 0, 1, 2, 3, 4, and 5 respectively. The Trace window contains a status of the trace. In this example, 1,445,448 trace records are used out of a total of 201,326,592 total record memory space in the TRACE32 serial trace tool.

To facilitate post-processing of the data, execute the following commands in a CMM script:

```
// ********************************************************************************
// Example T32 CMM script
// Script extracts data results from a T32 trace listing and displays
// those extract results in the AREA window
// ********************************************************************************
AREA.Create demowin 500. 500.
winpos 0% 0% 60% 70%
AREA.view demowin
AREA.Select demowin

Trace.Find CYcle wr-m01                          // find first write cycle
RePeat 0
(
  if !FOUND()
    ENDDO
  &record=TRACK.RECORD()                        // get record number
  &data_result=Analyzer.RECORD.DATA(&record)    // get data
  PRINT "DATA: " FORMAT.HEX(16.,&data_result)
  Trace.Find                                    // find next write cycle
)
```

Document Number AN5123
Revision 0, 05/2015