Synchronous Debug & Trace on two Infineon AURIX Devices

Publish Date: 03/12/2019

Table of Contents

1 winIDEA Workspace Configuration

For synchronous debug and trace of two AURIX devices, two winIDEA workspaces need to be created, a “master” workspace and a “slave” workspace. Which one of the two AURIX devices acts as a master or as slave, is, in terms of synchronous debug/trace operation, an arbitrary selection by the user.

In the following example, we assume that the “master” device uses an iC5700 + Active Probe Infineon AGBT. The “slave” device uses an iC5700 + Active Probe Infineon DAP.

In addition, each iC5700 is equipped with an FNET Hub and a CAN2/LIN2 Add-On module. Both iC5700 FNET Hubs are connected via the FNET SYNC cable.
Figure 1 depicts the overall setup.

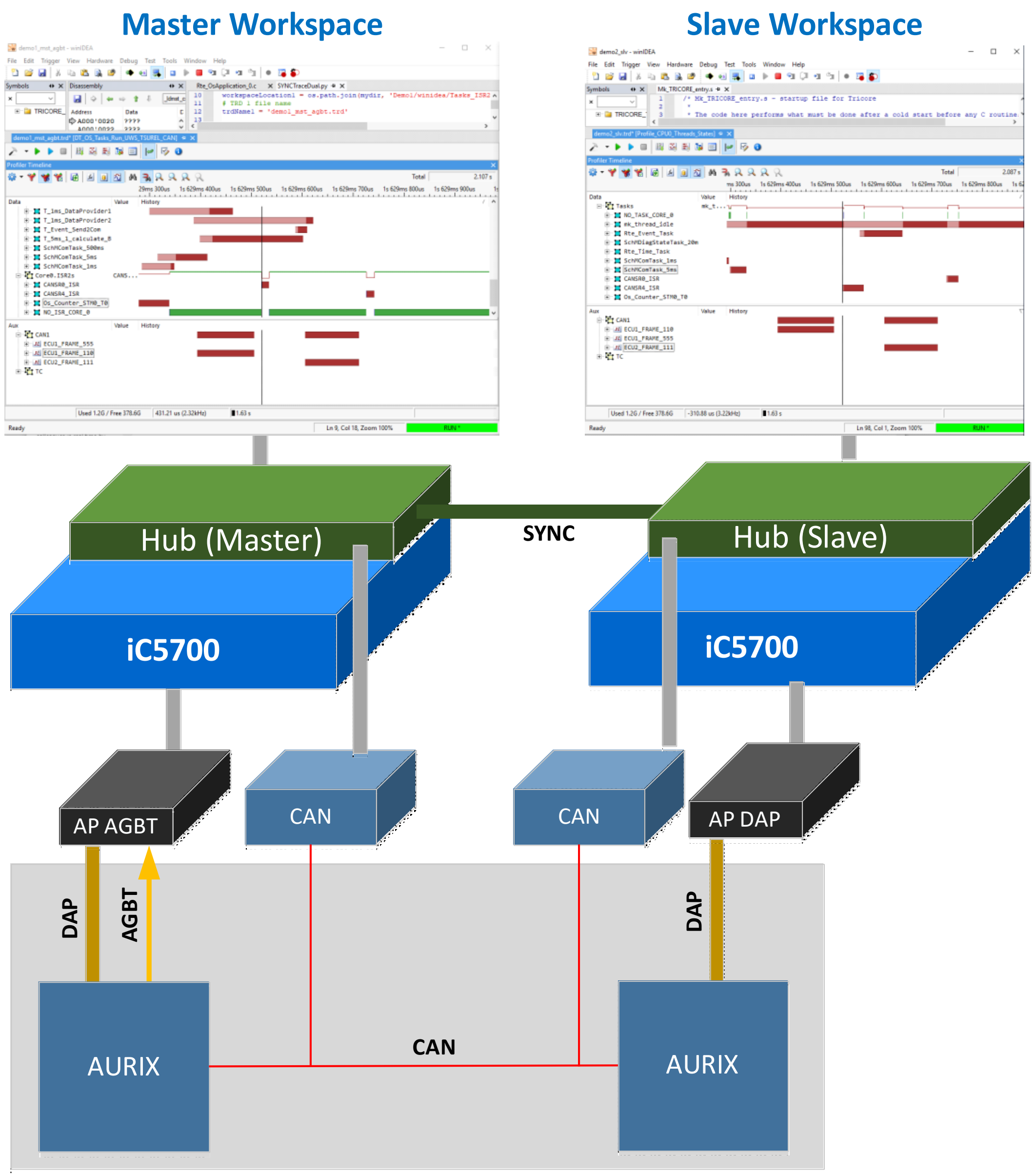


Figure 1: Sample Dual-AURIX Synchronous Debug & Trace Setup

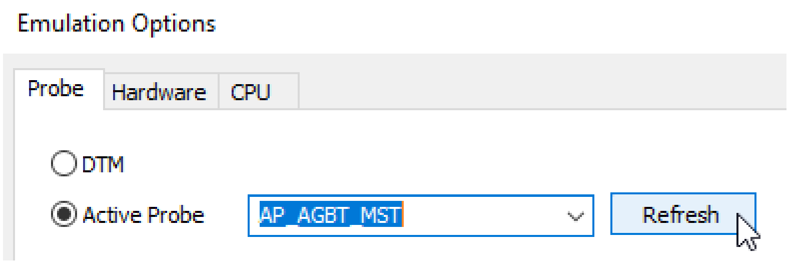
1.1 Master winIDEA Workspace Configuration

The overall winIDEA configuration starts with the Master winIDEA Instance.
Launch winIDEA and create a basic AURIX Debug and Trace workspace.

The following workspace configurations are relevant for sync debug and trace.

1.1.1 Infineon AGBT Active Probe Detection

After the communication to the iC570 has been established, it is recommended to perform a detection of the connected Active Probe. This can be done via the menu "Hardware – Emulation Options – Probe". Select Active Probe and then click the "Refresh" button. Select the detected AGBT Active Probe.

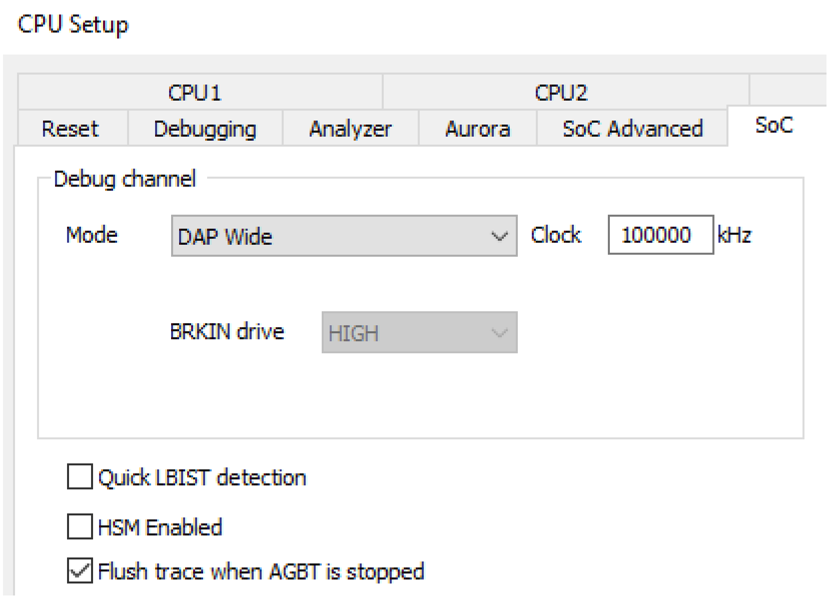


[✓ Active Probe] AP_AGBT_MST

Figure 2: AGBT Active Probe Detection

1.1.2 DAP Clock Frequency

The DAP clock frequency can be set in the "CPU Setup - SoC" dialog. This dialog can be opened via the menu "Hardware – CPU Options..."
The DAP clock frequency should be set to a high clock rate for an optimized the target status polling rate. A high target polling rate improves the sync debug latencies as well as the time synchronization between on-chip timestamp and BlueBox timestamp.
In Figure 3 the DAP clock is set to 100MHz. The maximum possible clock rate mainly depends on the DAP signal routing on the target board.



Mode: DAP Wide
Clock: 100000 kHz
BRKIN drive: HIGH

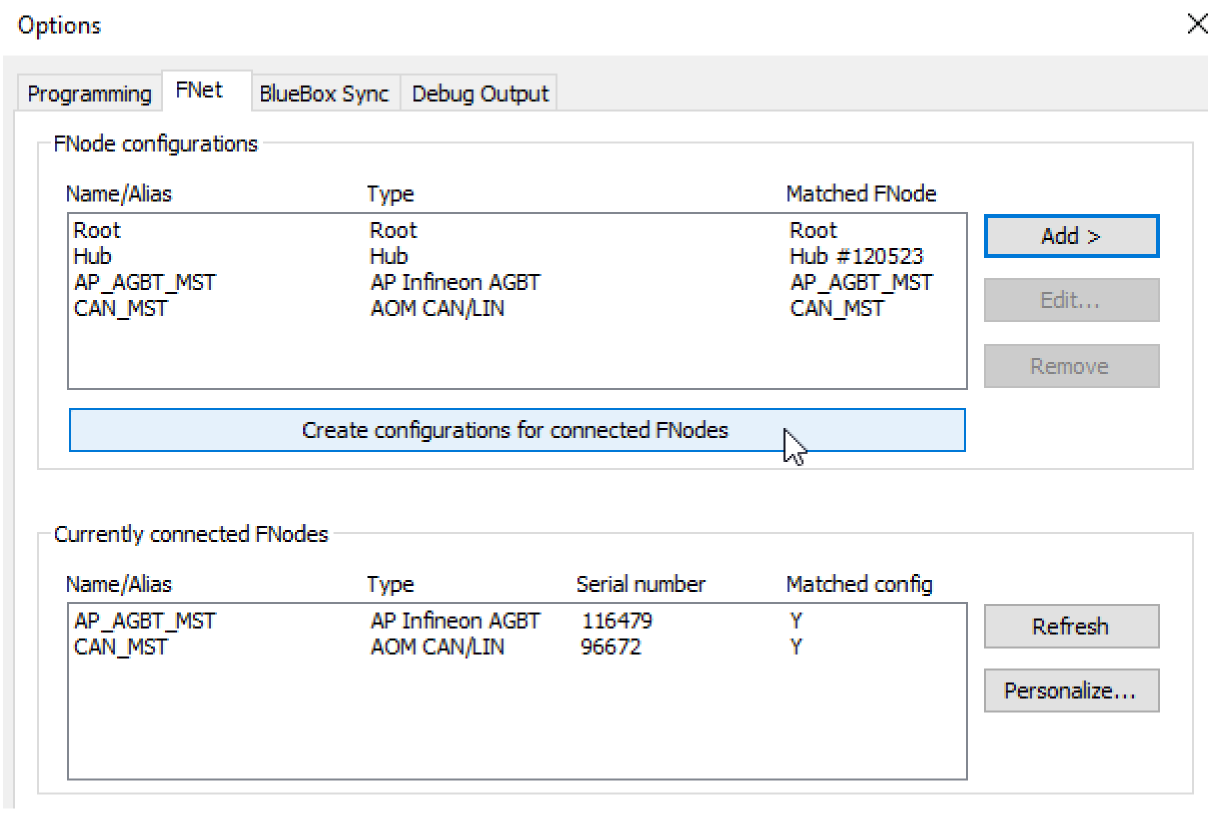
- [ ] Quick LBIST detection
- [ ] HSM Enabled
- [x] Flush trace when AGBT is stopped

Figure 3: DAP Clock Configuration

1.1.3 FNET Configuration

For synchronized debug/trace, at least an Active Probe (either DAP or AGBT) needs to be connected to the iC5700 via FNET. The Active Probes should be connected to the FNET connector on the front-side of the iC5700 Base Unit. Optionally, other Add-On modules can be connected to the FNET connectors of the iC5700 Hub.
The complete FNET configuration of an iC5700 can be set in the “Options – FNet” dialog, which can be opened via the “Hardware – Options…” menu.

A good start for creating a new configuration is to push the “Refesh” button on the “Currently connected FNodes” section. This updates the list of currently connected FNodes,i.e. Active Probes and AOMs.
From this list either individual FNodes can be included the “FNode configuration” via the “Add >” button or all connected FNodes can be added by clicking the “Create configuration for connected FNodes” button.



Name/Alias	Type	Matched FNode
Root	Root	Root
Hub	Hub	Hub #120523
AP_AGBT_MST	AP Infineon AGBT	AP_AGBT_MST
CAN_MST	AOM CAN/LIN	CAN_MST

Name/Alias	Type	Serial number	Matched config
AP_AGBT_MST	AP Infineon AGBT	116479	Y
CAN_MST	AOM CAN/LIN	96672	Y

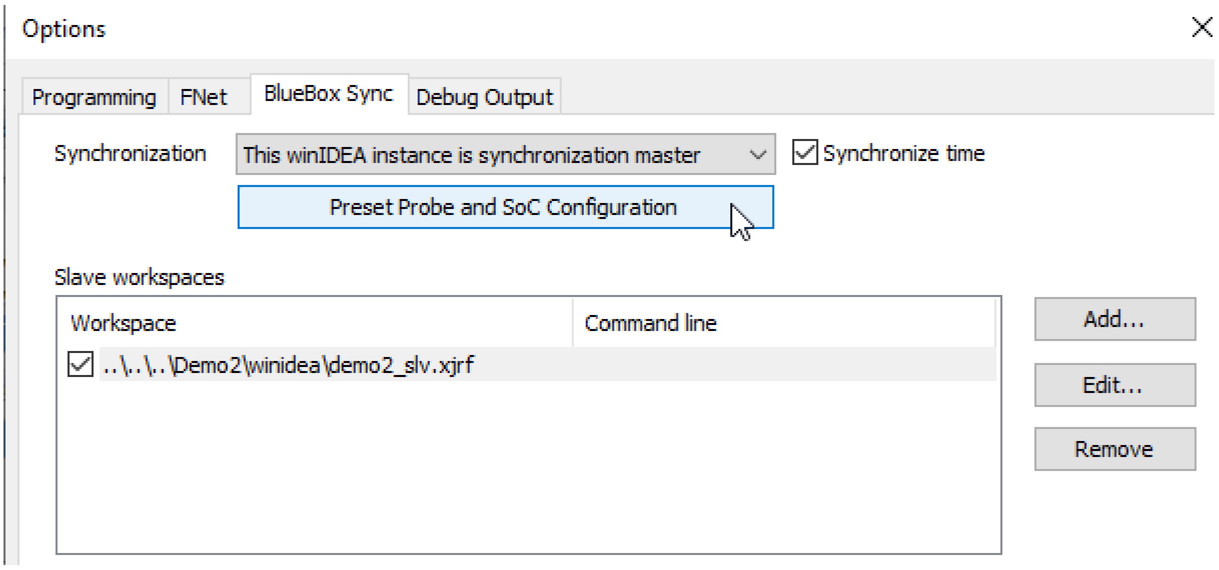
Figure 4: Sample FNET Configuration (Master winIDEA Instance using AGBT Active Probe and CAN/LIN AOM)

1.1.4 BlueBox Synchronization

Next step is to establish the master-slave connection between the two winIDEA workspaces, the master and the slave workspace. This is done in the “Options – BlueBox Sync” dialog, opened via the “Hardware – Options…” menu.
Figure 5 shows a sample BlueBox Sync configuration of a Master winIDEA instance. This winIDEA instance is selected to the Master by choosing “This winIDEA instance is synchronization master”.
The winIDEA workspace of the Slave winIDEA instance can be selected via “Slave workspaces - Add…”.

Pushing the "Preset Probe and SoC Configuration" causes winIDEA to automatically adjust the Active Probe and SoC configurations needed for synchronized debug and trace.

The "Synchronize time" tick box must be enabled for synchronized trace.



Options

Programming | FNet | BlueBox Sync | Debug Output

Synchronization: This winIDEA instance is synchronization master

☑ Synchronize time

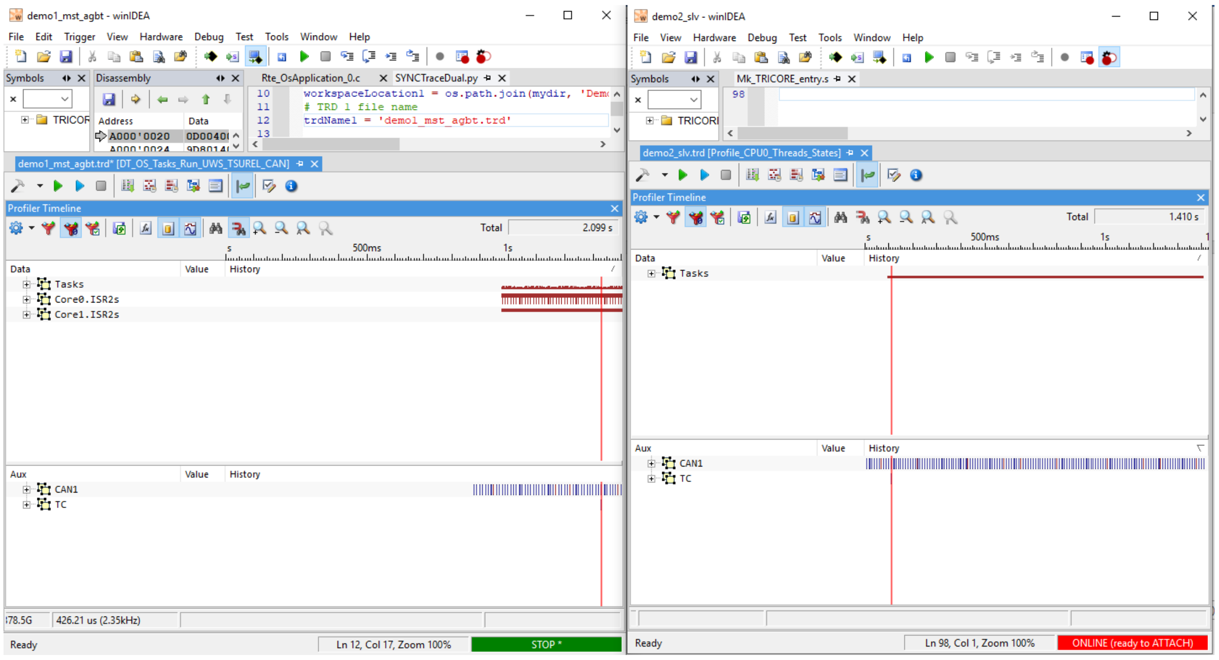
Preset Probe and SoC Configuration

Slave workspaces

Workspace	Command line
☑ ..\..\..\Demo2\winidea\demo2_slv.xjrf	

Figure 5: Sample Master winIDEA BlueBox Sync Dialog

The above settings become active in the iC5700, the Active Probe, AOMs and on the target, by performing a Download (menu: »Debug – Download«) or Symbol Download (menu: »Debug – Load Symbols only«).
The download operation also launches the Slave winIDEA Instance.
After successful download and lauch, the Master winIDEA instance should be in state »**STOP***«, the Slave winIDEA instance in state »**ONLINE (ready to ATTACH)**«.
The '*' in the winIDEA status indicates the synchronized debug is not now active.
Figure 6 depicts a Master and Slave winIDEA instance after a successful download operation in the Master winIDEA instance.



demo1_mst_agbt - winIDEA

```
workspaceLocation1 = os.path.join(mydir, 'Dem
# TRD 1 file name
trdName1 = 'demo1_mst_agbt.trd'
```

demo1_mst_agbt.trd* [DT_OS_Tasks_Run_UWS_TSUREL_CAN]

Profiler Timeline — Total 2.099 s

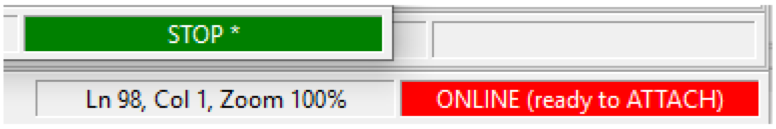
Ready | Ln 12, Col 17, Zoom 100% | STOP *

demo2_slv - winIDEA

demo2_slv.trd [Profile_CPU0_Threads_States]

Profiler Timeline — Total 1.410 s

Ready | Ln 98, Col 1, Zoom 100% | ONLINE (ready to ATTACH)


STOP *

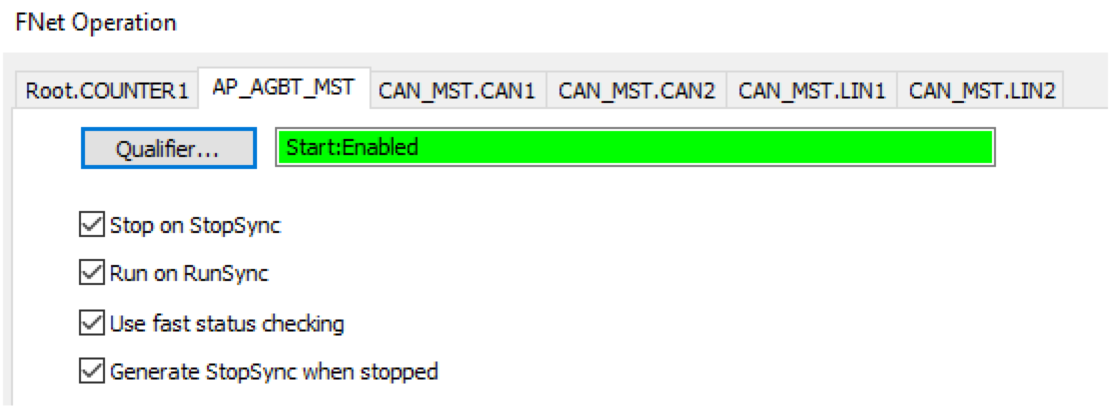
Ln 98, Col 1, Zoom 100%

ONLINE (ready to ATTACH)

Figure 6: Sample Master and Slave winIDEA Instances after a Download Operation in the Master Instance.

1.1.5 FNET Operation

Finally, the “FNET Operation” setting for the Active Probes should be checked.
Open the “FNET Operation” dialog via the menu “Hardware – FET Operation...”.
As shown in Figure 7, the “Qualifier...” should be set to “Start:Enabled”. In addition, all four tick boxes for “Stop on StopSync”, “Run on RunSync”, “Use fast status polling” and “Generate StopSync when stopped” must be enabled to allow for synchronized debug.


FNet Operation

Root.COUNTER1 | AP_AGBT_MST | CAN_MST.CAN1 | CAN_MST.CAN2 | CAN_MST.LIN1 | CAN_MST.LIN2

Qualifier... Start:Enabled

☑ Stop on StopSync

☑ Run on RunSync

☑ Use fast status checking

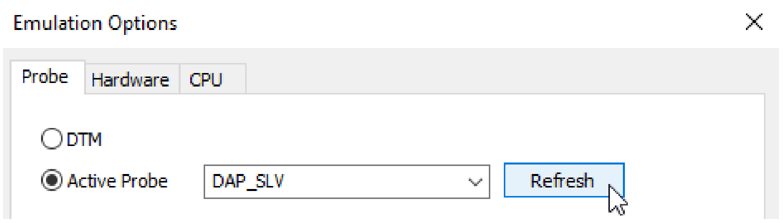
☑ Generate StopSync when stopped

Figure 7: FNET Operation Dialog for an AGBT Active Probe supporting Synchronized Debug

1.2 Slave winIDEA Workspace Configuration

1.2.1 DAP Active Probe Detection

After the communication to the iC570 has been established, it is recommended to perform a detection of the connected Active Probe. This can be done via the menu “Hardware – Emulation Options – Probe”. Select Active Probe and then click the “Refresh” button. Select the detected DAP Active Probe. In the Active Probe detection shown in Figure 8, the Active Probe has been given a alias “DAP_SLV”.

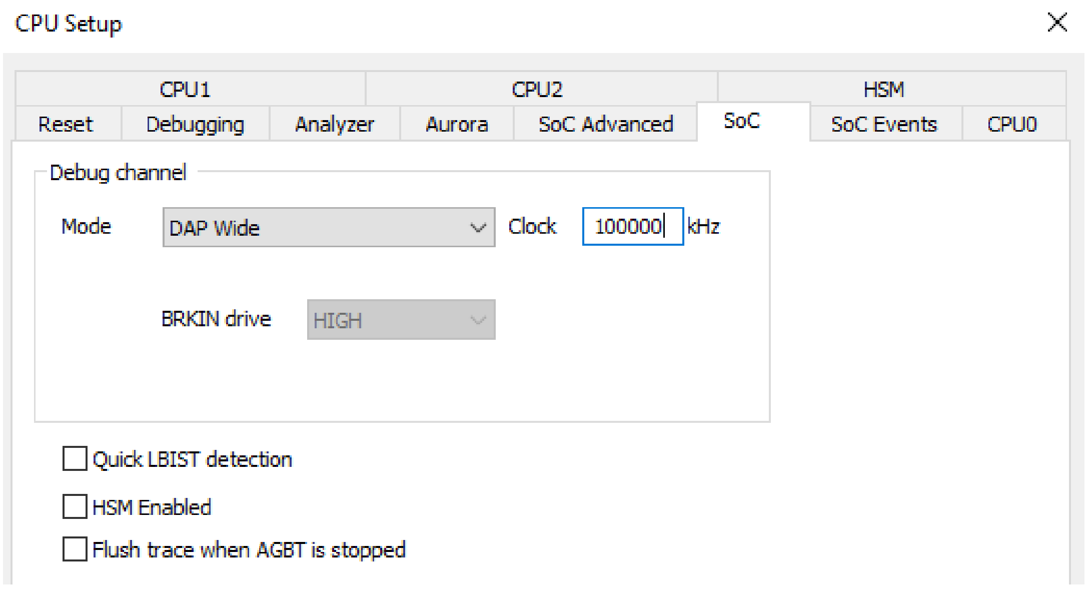


DTM
[✓ Active Probe] DAP_SLV

Figure 8: iC5700 Active Probe Detection.

1.2.2 DAP Clock Frequency

The DAP clock frequency can be set in the “CPU Setup - SoC” dialog. This dialog can be opened via the menu “Hardware – CPU Options...”
The DAP clock frequency should be set to a high clock rate for an optimized the target status polling rate. A high target polling rate improves the sync debug latencies as well as the time synchronization between on-chip timestamp and BlueBox timestamp.
In Figure 9 the DAP clock is set to 100MHz. The maximum possible clock rate mainly depends on the DAP signal routing on the target board.



Mode: DAP Wide
Clock: 100000 kHz
BRKIN drive: HIGH
☐ Quick LBIST detection
☐ HSM Enabled
☐ Flush trace when AGBT is stopped

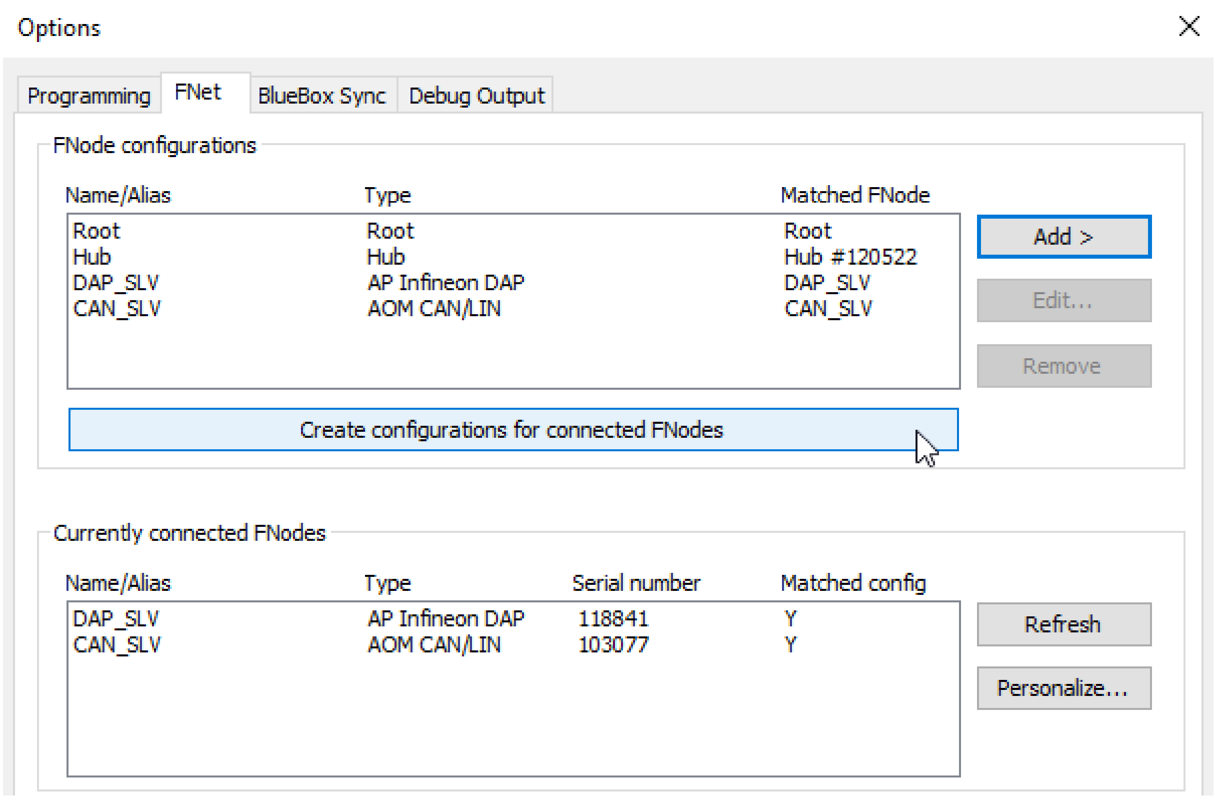
Figure 9: DAP Clock Configuration

1.2.3 FNET Configuration

For synchronized debug/trace, at least an Active Probe (either DAP or AGBT) needs to be connected to the iC5700 via FNET. The Active Probes should be connected to the FNET connector on the front-side of the iC5700 Base Unit. Optionally, other Add-On modules can be connected to the FNET connectors of the iC5700 Hub.

The complete FNET configuration of an iC5700 can be set in the “Options – FNet” dialog, which can be opened via the “Hardware – Options…” menu.

A good start for creating a new configuration is to push the “Refesh” button on the “Currently connected FNodes” section. This updates the list of currently connected FNodes,i.e. Active Probes and AOMs.
From this list either individual FNodes can be included the “FNode configuration” via the “Add >” button or all connected FNodes can be added by clicking the “Create configuration for connected FNodes” button.



Name/Alias	Type	Matched FNode
Root	Root	Root
Hub	Hub	Hub #120522
DAP_SLV	AP Infineon DAP	DAP_SLV
CAN_SLV	AOM CAN/LIN	CAN_SLV

Name/Alias	Type	Serial number	Matched config
DAP_SLV	AP Infineon DAP	118841	Y
CAN_SLV	AOM CAN/LIN	103077	Y

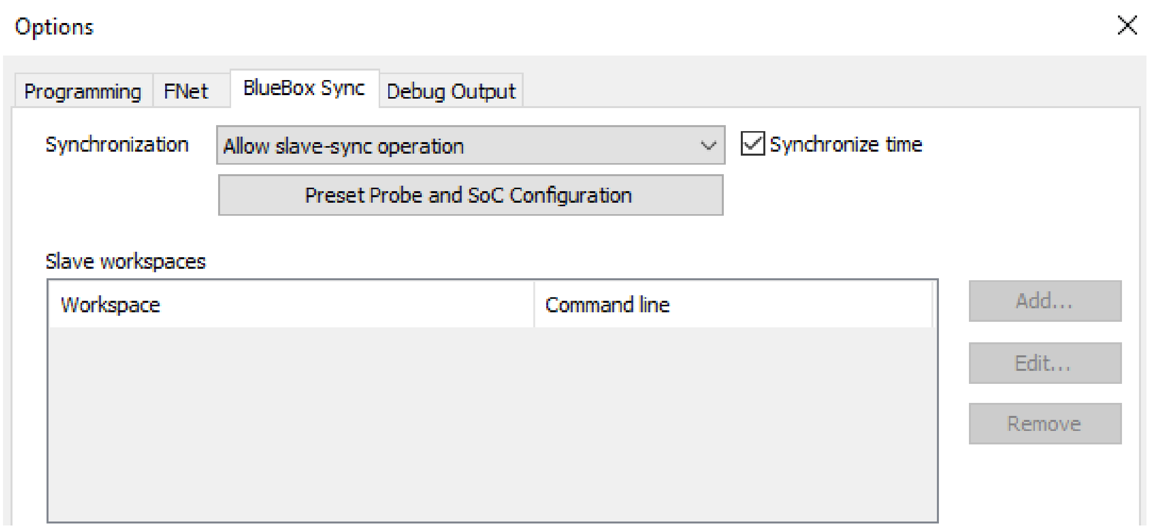
Figure 10: Sample FNET Configuration (Slave winIDEA Instance using DAP Active Probe and CAN/LIN AOM)

1.2.4 BlueBox Synchronization

Next step is to establish the master-slave connection between the two winIDEA workspaces, the master and the slave workspace. This is done in the “Options – BlueBox Sync” dialog, opened via the “Hardware – Options…” menu.
Figure 11 shows a sample BlueBox Sync configuration of a Slave winIDEA instance. This winIDEA instance is selected to the Slave by choosing “Allow slave-sync operation”.

Pushing the “Preset Probe and SoC Configuration” causes winIDEA to automatically adjust the Active Probe and SoC configurations needed for synchronized debug and trace.

The “Synchronize time” tick box must be enabled for synchronized trace.



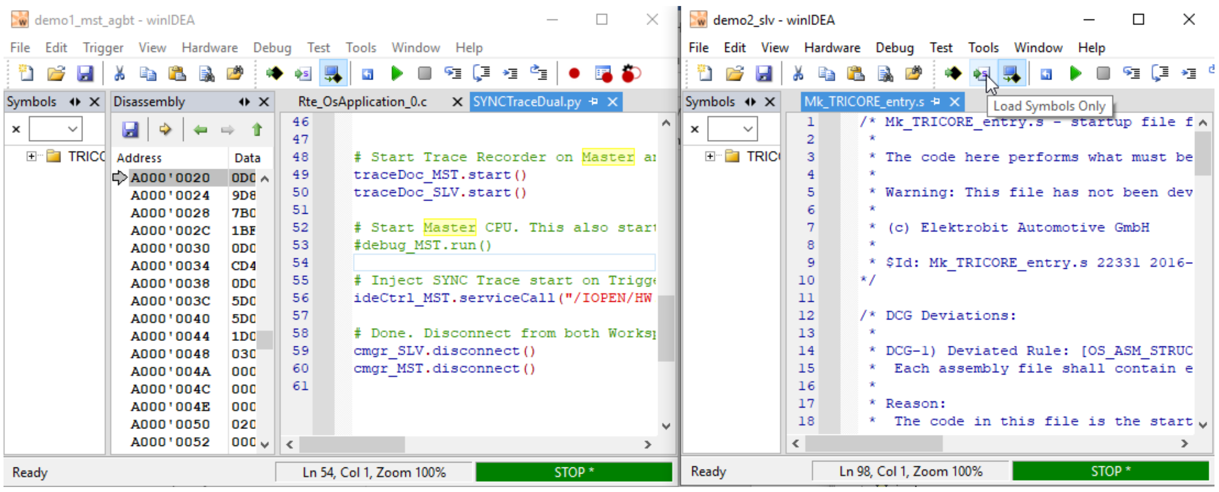
Synchronization: Allow slave-sync operation
[✓] Synchronize time

Slave workspaces

Workspace	Command line

Figure 11: Sample Slave winIDEA BlueBox Sync Dialog

The above settings become active in the iC5700, the Active Probe, AOMs and on the target, by performing a Download (menu: »Debug – Download«) or Symbol Download (menu: »Debug – Load Symbols only«).
After successful download and lauch, the Slave winIDEA instance should be in state »**STOP***« (same as Master winIDEA instance).
The '*' in the winIDEA status indicates the synchronized debug is not now active.
Figure 12 depicts a Master and Slave winIDEA instance after a successful download operation in the Slave winIDEA instance.



```
Address     Data
A000'0020   0DC
A000'0024   9D8
A000'0028   7BC
A000'002C   1BF
A000'0030   0DC
A000'0034   CD4
A000'0038   0DC
A000'003C   5DC
A000'0040   5DC
A000'0044   1DC
A000'0048   03C
A000'004A   00C
A000'004C   00C
A000'004E   00C
A000'0050   02C
A000'0052   00C
```

```
# Start Trace Recorder on Master ar
traceDoc_MST.start()
traceDoc_SLV.start()

# Start Master CPU. This also start
#debug_MST.run()

# Inject SYNC Trace start on Trigge
ideCtrl_MST.serviceCall("/IOPEN/HW

# Done. Disconnect from both Worksp
cmgr_SLV.disconnect()
cmgr_MST.disconnect()
```

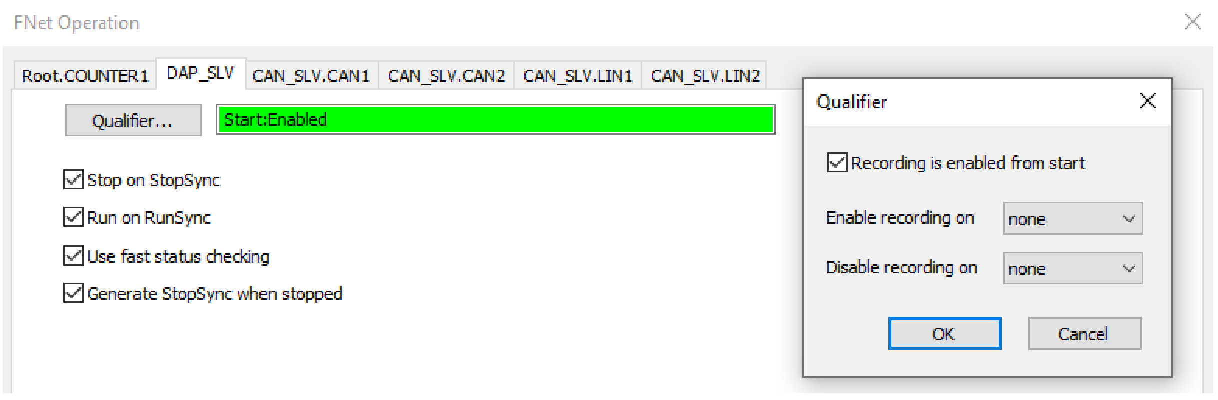
```
/* Mk_TRICORE_entry.s - startup file f
 *
 * The code here performs what must be
 *
 * Warning: This file has not been dev
 *
 * (c) Elektrobit Automotive GmbH
 *
 * $Id: Mk_TRICORE_entry.s 22331 2016-
*/

/* DCG Deviations:
 *
 * DCG-1) Deviated Rule: [OS_ASM_STRUC
 *  Each assembly file shall contain e
 *
 * Reason:
 *  The code in this file is the start
```

Ready | Ln 54, Col 1, Zoom 100% | STOP *
Ready | Ln 98, Col 1, Zoom 100% | STOP *

Figure 12: Sample Master and Slave winIDEA Instances after a Download Operation in the Slave Instance.

1.2.5 FNET Operation

Finally, the “FNET Operation” setting for the Active Probes should be checked.
Open the “FNET Operation” dialog via the menu “Hardware – FET Operation…”.
As shown in Figure 7, the “Qualifier…” should be set to “Start:Enabled”. In addition, all four tick boxes for “Stop on StopSync”, “Run on RunSync”, “Use fast status polling” and “Generate StopSync when stopped” must be enabled to allow for synchronized debug.



Start:Enabled

☑ Stop on StopSync
☑ Run on RunSync
☑ Use fast status checking
☑ Generate StopSync when stopped

☑ Recording is enabled from start
Enable recording on: none
Disable recording on: none

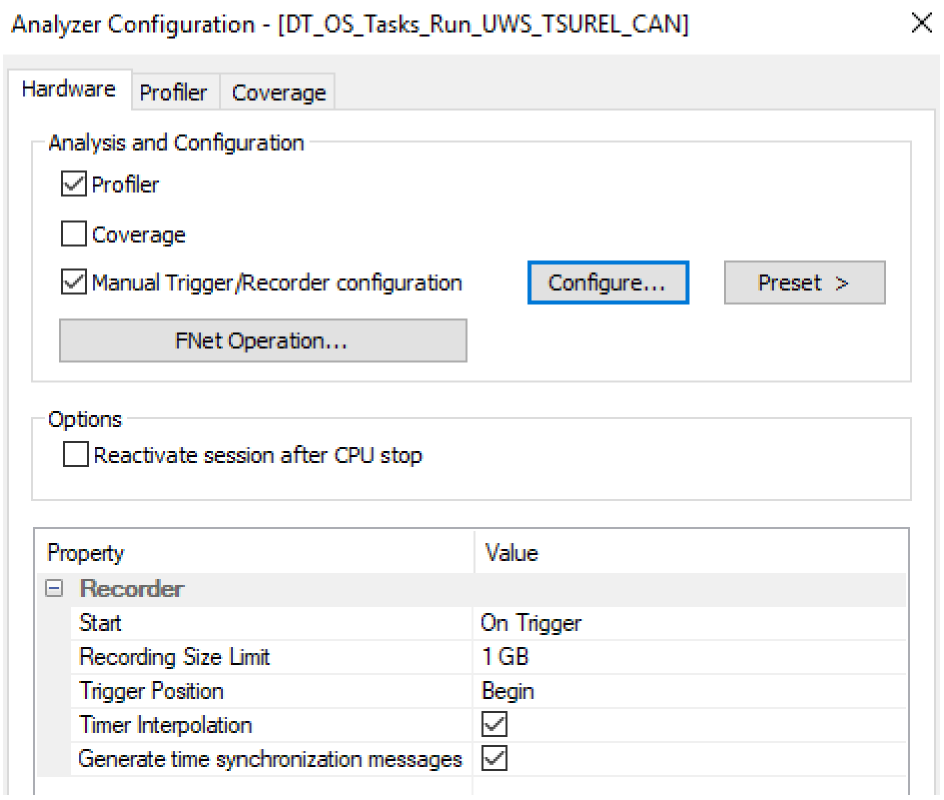
Figure 13: FNET Operation Dialog for a DAP Active Probe supporting Synchronized Debug

2 winIDEA Analyzer Configuration for Synchronized Trace

2.1 Recorder (i.e. iC5700) Configuration

As depicted in Figure 14 and Figure 15, the Analyzer should enable “Profiler” and “Manual Trigger/Recorder configuration”.
In addition, the iC5700 trace recorder must be set to:

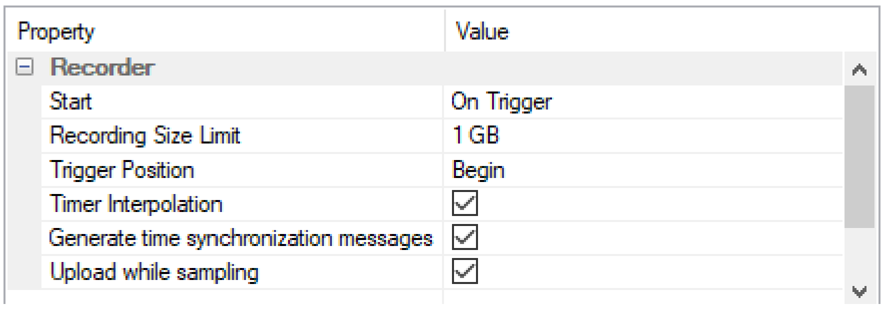
- Start: On Trigger
- Timer Interpolation: enabled
- Generate time synchronization messages: enabled
- (for DAP) Upload while sampling: enabled


Analyzer Configuration - [DT_OS_Tasks_Run_UWS_TSUREL_CAN]
Hardware
Profiler
Coverage
Analysis and Configuration
Configure...
Preset >
FNet Operation...
Options

[✓] Profiler
[ ] Coverage
[✓] Manual Trigger/Recorder configuration
[ ] Reactivate session after CPU stop

Property	Value
Recorder	
Start	On Trigger
Recording Size Limit	1 GB
Trigger Position	Begin
Timer Interpolation	✓
Generate time synchronization messages	✓

Figure 14: iC5700 Recorder Configuration for Synchronized Trace using AGBT Active Probe



Property	Value
Recorder	
Start	On Trigger
Recording Size Limit	1 GB
Trigger Position	Begin
Timer Interpolation	✓
Generate time synchronization messages	✓
Upload while sampling	✓

Figure 15: iC5700 Recorder Configuration for Synchronized Trace using DAP Active Probe

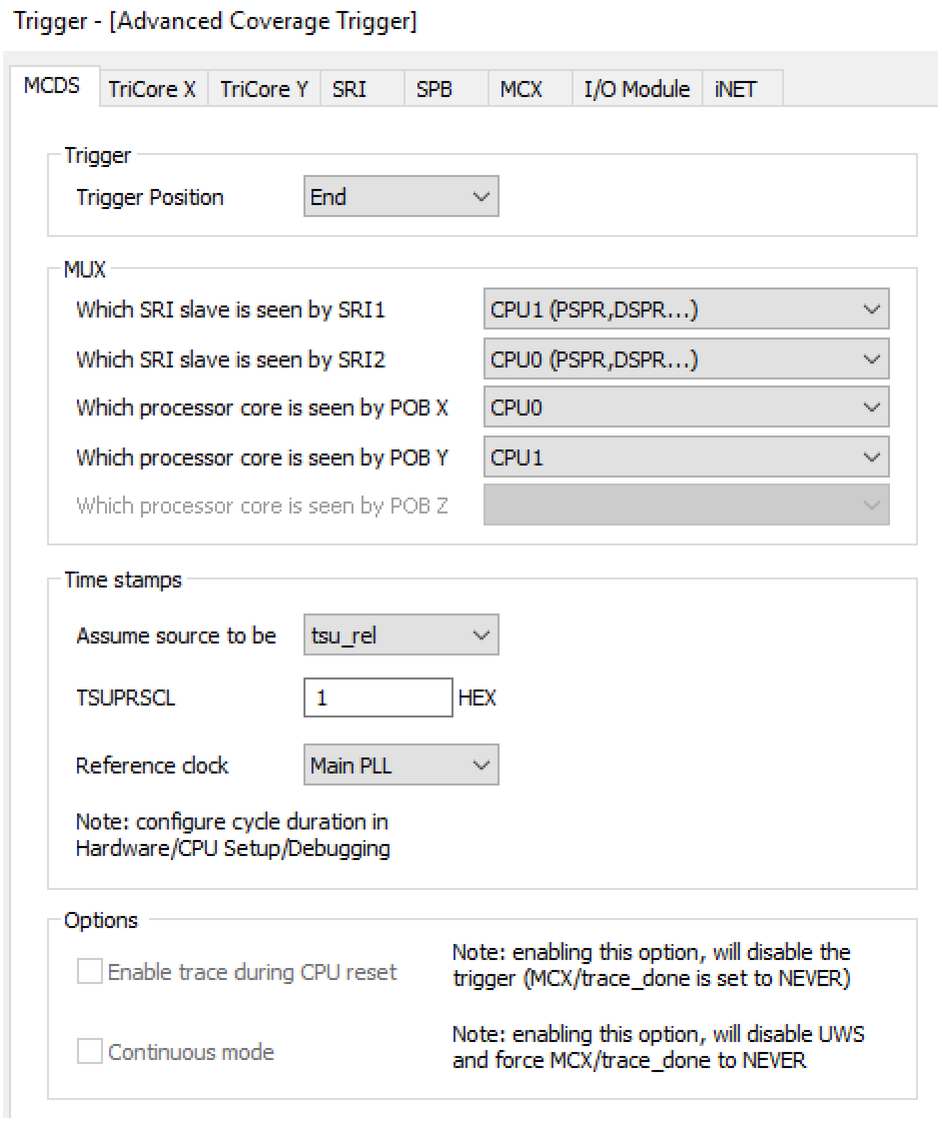
2.2 AURIX-specific Manual Trace Configuration (i.e. MCDS Configuration)

Trigger Position:
As the Emulation Memory of the AURIX is used as FIFO for the trace data streaming (Upload-While-Sampling) to the iC5700, the Trigger Position can be set to »End«.

MUX:
The POB and BOB Muxes must be set as appropriate for the individual trace use-case.

Time stamps:
The time stamp source must be »tsu_rel«.
The TSU_REL Prescaler value (TSUPRSCL) may be chosen as appropriate for the individual trace use-case. A TSUPRSCL value of 1 yields the maximum trace timing accuracy.
NOTE: Synchronized trace only works when using TSU_REL time stamping. TICK time stamping is not supported for synchronized trace.

Figure 16 shows a sample MCDS configuration supporting synchronized trace.



Trigger
Trigger Position: End

MUX
Which SRI slave is seen by SRI1: CPU1 (PSPR,DSPR...)
Which SRI slave is seen by SRI2: CPU0 (PSPR,DSPR...)
Which processor core is seen by POB X: CPU0
Which processor core is seen by POB Y: CPU1
Which processor core is seen by POB Z:

Time stamps
Assume source to be: tsu_rel
TSUPRSCL: 1 HEX
Reference clock: Main PLL
Note: configure cycle duration in Hardware/CPU Setup/Debugging

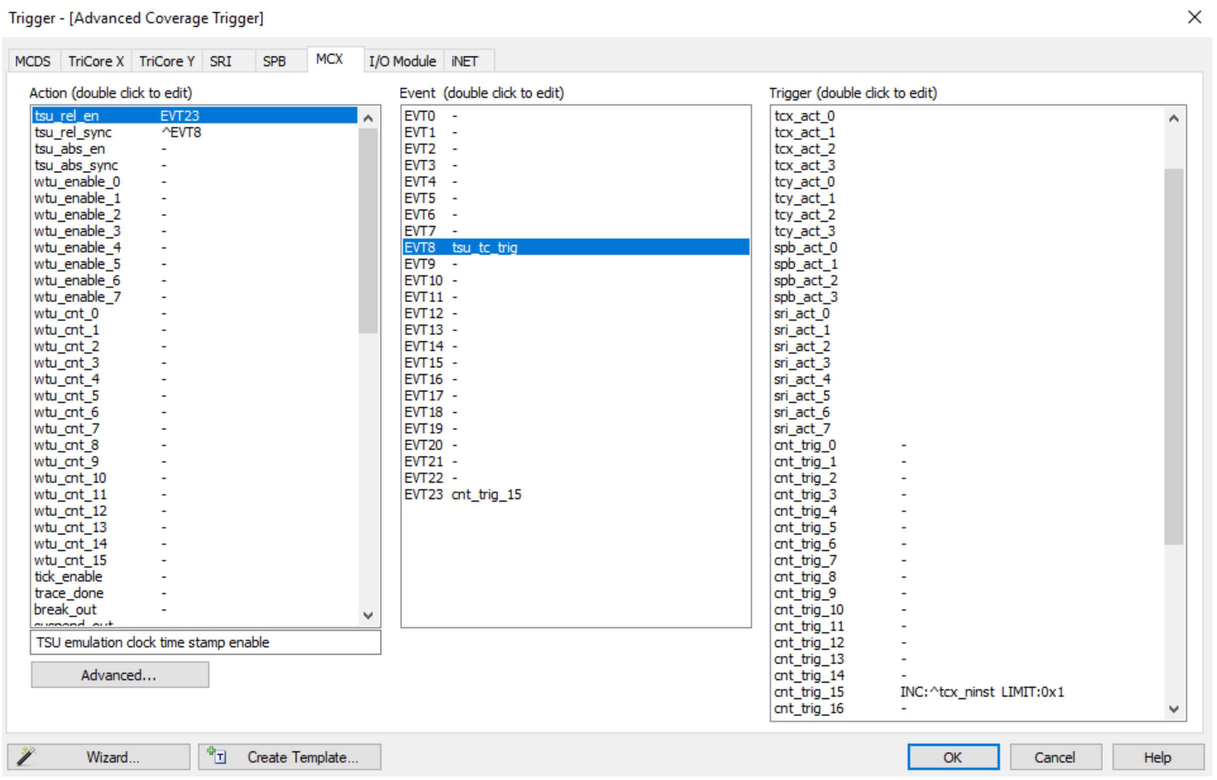
Options
☐ Enable trace during CPU reset — Note: enabling this option, will disable the trigger (MCX/trace_done is set to NEVER)
☐ Continuous mode — Note: enabling this option, will disable UWS and force MCX/trace_done to NEVER

Figure 16_ Sample MCDS Configuration supporting synchronized Trace

MCDS Multi-Core Cross Connect (MCX) Configuration

The MCX module of MCDS must be configured for TSU_REL time stamping.

In the sample MCX configuration shown in Figure 17, the MCX counter 15 in conjuction with EVT23 is used to enable TSU_REL time stamp messaging only while the core (attached to POB X) is executing instructions (i.e. is running). EVT8 is connected the tsu_tc_trig (i.e. to the TSUPRSCL prescaler). This means, a TSU_REL time stamp sync message is generated periodically upon every underflow of the TSUPRSCL prescaler.
In addition, the MCX action »trace_done« is set to NEVER, which is requird for »Upload-While-Sampling« mode.



Action (double click to edit)	
tsu_rel_en	EVT23
tsu_rel_sync	^EVT8
tsu_abs_en	-
tsu_abs_sync	-
wtu_enable_0	-
wtu_enable_1	-
wtu_enable_2	-
wtu_enable_3	-
wtu_enable_4	-
wtu_enable_5	-
wtu_enable_6	-
wtu_enable_7	-
wtu_cnt_0	-
wtu_cnt_1	-
wtu_cnt_2	-
wtu_cnt_3	-
wtu_cnt_4	-
wtu_cnt_5	-
wtu_cnt_6	-
wtu_cnt_7	-
wtu_cnt_8	-
wtu_cnt_9	-
wtu_cnt_10	-
wtu_cnt_11	-
wtu_cnt_12	-
wtu_cnt_13	-
wtu_cnt_14	-
wtu_cnt_15	-
tick_enable	-
trace_done	-
break_out	-

TSU emulation clock time stamp enable

Event (double click to edit)	
EVT0	-
EVT1	-
EVT2	-
EVT3	-
EVT4	-
EVT5	-
EVT6	-
EVT7	-
EVT8	tsu_tc_trig
EVT9	-
EVT10	-
EVT11	-
EVT12	-
EVT13	-
EVT14	-
EVT15	-
EVT16	-
EVT17	-
EVT18	-
EVT19	-
EVT20	-
EVT21	-
EVT22	-
EVT23	cnt_trig_15

Trigger (double click to edit)	
tcx_act_0	
tcx_act_1	
tcx_act_2	
tcx_act_3	
tcy_act_0	
tcy_act_1	
tcy_act_2	
tcy_act_3	
spb_act_0	
spb_act_1	
spb_act_2	
spb_act_3	
sri_act_0	
sri_act_1	
sri_act_2	
sri_act_3	
sri_act_4	
sri_act_5	
sri_act_6	
sri_act_7	
cnt_trig_0	-
cnt_trig_1	-
cnt_trig_2	-
cnt_trig_3	-
cnt_trig_4	-
cnt_trig_5	-
cnt_trig_6	-
cnt_trig_7	-
cnt_trig_8	-
cnt_trig_9	-
cnt_trig_10	-
cnt_trig_11	-
cnt_trig_12	-
cnt_trig_13	-
cnt_trig_14	-
cnt_trig_15	INC:^tcx_ninst LIMIT:0x1
cnt_trig_16	-

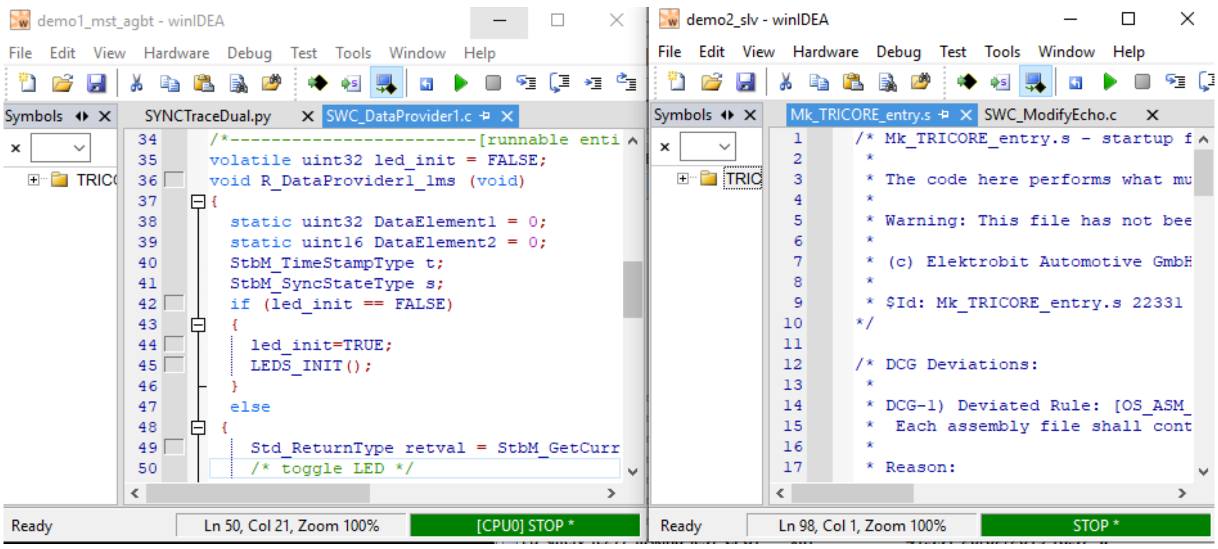
Figure 17: Sample MCX Configuration using TSU_REL time stamping required for Synchronized Trace

3 Synchronized Debug Operation

This section describes the basic synchronized debug operations.

3.1 Initial State after Download

After a download operation on both Master and Slave winIDEA instance, both devices are in »**STOP***« state (see Figure 18). The '*' indicates that synchronized debug is currently active.



```
/*------------------------[runnable enti
volatile uint32 led_init = FALSE;
void R_DataProvider1_1ms (void)
{
  static uint32 DataElement1 = 0;
  static uint16 DataElement2 = 0;
  StbM_TimeStampType t;
  StbM_SyncStateType s;
  if (led_init == FALSE)
  {
    led_init=TRUE;
    LEDS_INIT();
  }
  else
  {
    Std_ReturnType retval = StbM_GetCurr
    /* toggle LED */
```

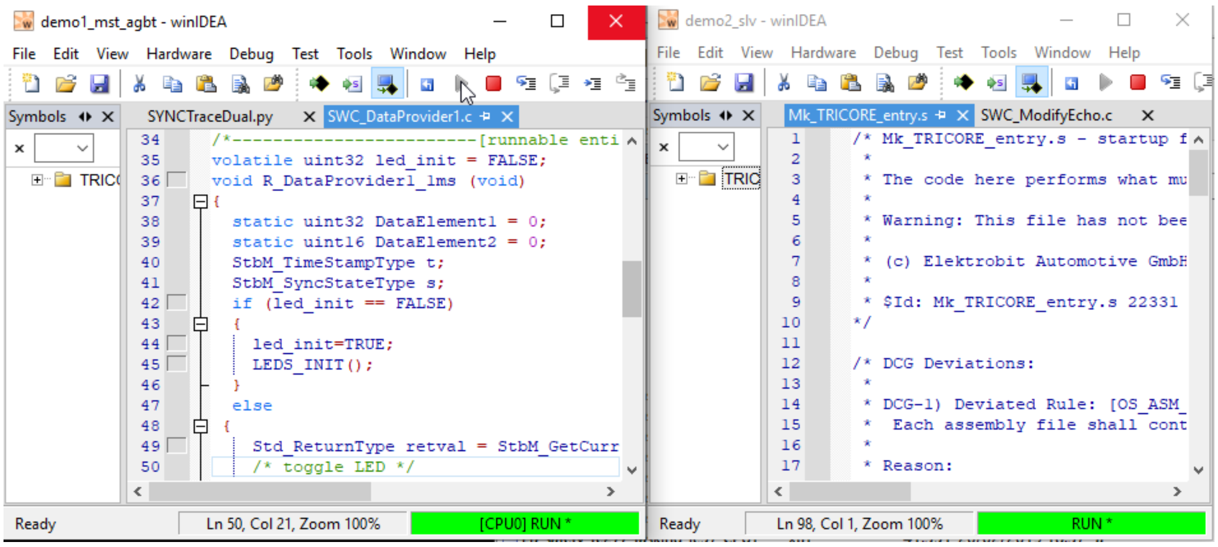
```
/* Mk_TRICORE_entry.s - startup f
 *
 * The code here performs what mu
 *
 * Warning: This file has not bee
 *
 * (c) Elektrobit Automotive GmbH
 *
 * $Id: Mk_TRICORE_entry.s 22331
*/

/* DCG Deviations:
 *
 * DCG-1) Deviated Rule: [OS_ASM_
 *  Each assembly file shall cont
 *
 * Reason:
```

```
Ready    Ln 50, Col 21, Zoom 100%    [CPU0] STOP *
Ready    Ln 98, Col 1, Zoom 100%     STOP *
```

Figure 18: After download operation on both Master and Slave, both are in “STOP*” State.

3.2 Run / Stop / Breakpoint Operation

Starting the CPU (»Debug – Run Control – Run (F5)«) on the Master, also causes the Slave to enter »**RUN***« state (see Figure 19).



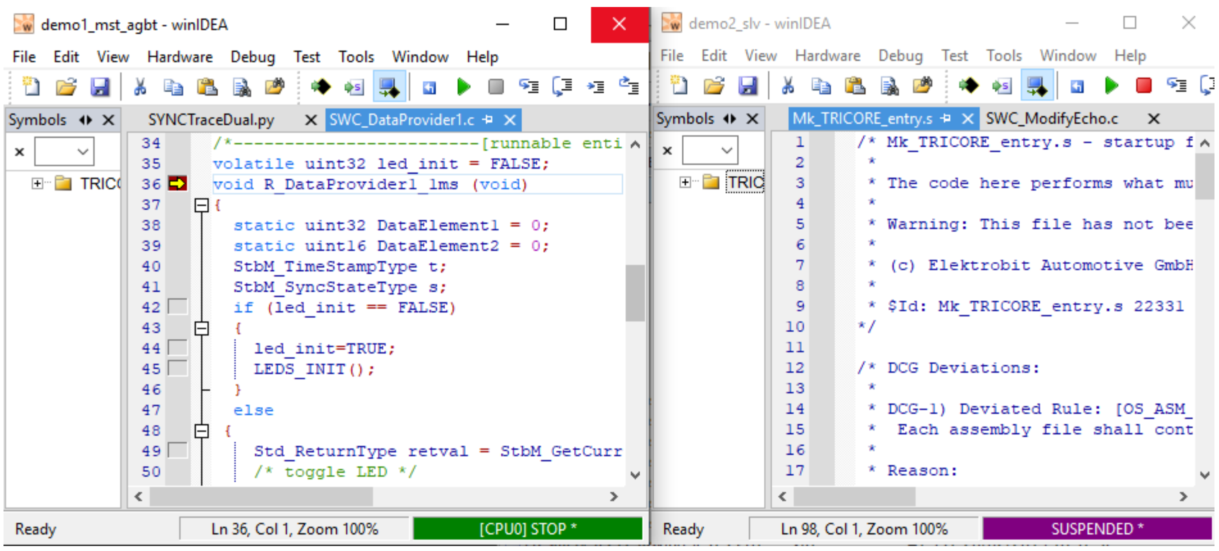
```
/*------------------------[runnable enti
volatile uint32 led_init = FALSE;
void R_DataProvider1_1ms (void)
{
  static uint32 DataElement1 = 0;
  static uint16 DataElement2 = 0;
  StbM_TimeStampType t;
  StbM_SyncStateType s;
  if (led_init == FALSE)
  {
    led_init=TRUE;
    LEDS_INIT();
  }
  else
  {
    Std_ReturnType retval = StbM_GetCurr
    /* toggle LED */
```

```
/* Mk_TRICORE_entry.s - startup f
 *
 * The code here performs what mu
 *
 * Warning: This file has not bee
 *
 * (c) Elektrobit Automotive GmbH
 *
 * $Id: Mk_TRICORE_entry.s 22331
*/

/* DCG Deviations:
 *
 * DCG-1) Deviated Rule: [OS_ASM_
 *  Each assembly file shall cont
 *
 * Reason:
```

```
Ready    Ln 50, Col 21, Zoom 100%    [CPU0] RUN *
Ready    Ln 98, Col 1, Zoom 100%     RUN *
```

Figure 19: A Run Operation on the Master also causes the Slave to enter “RUN*” State.

A Breakpoint hit on the Master also causes the Slave to halt. The CPU entes the AURIX-specfic »**SUSPENDED***« state (see Figure 20).



```
/*-----------------------[runnable enti
volatile uint32 led_init = FALSE;
void R_DataProvider1_1ms (void)
{
  static uint32 DataElement1 = 0;
  static uint16 DataElement2 = 0;
  StbM_TimeStampType t;
  StbM_SyncStateType s;
  if (led_init == FALSE)
  {
    led_init=TRUE;
    LEDS_INIT();
  }
  else
  {
    Std_ReturnType retval = StbM_GetCurr
    /* toggle LED */
```

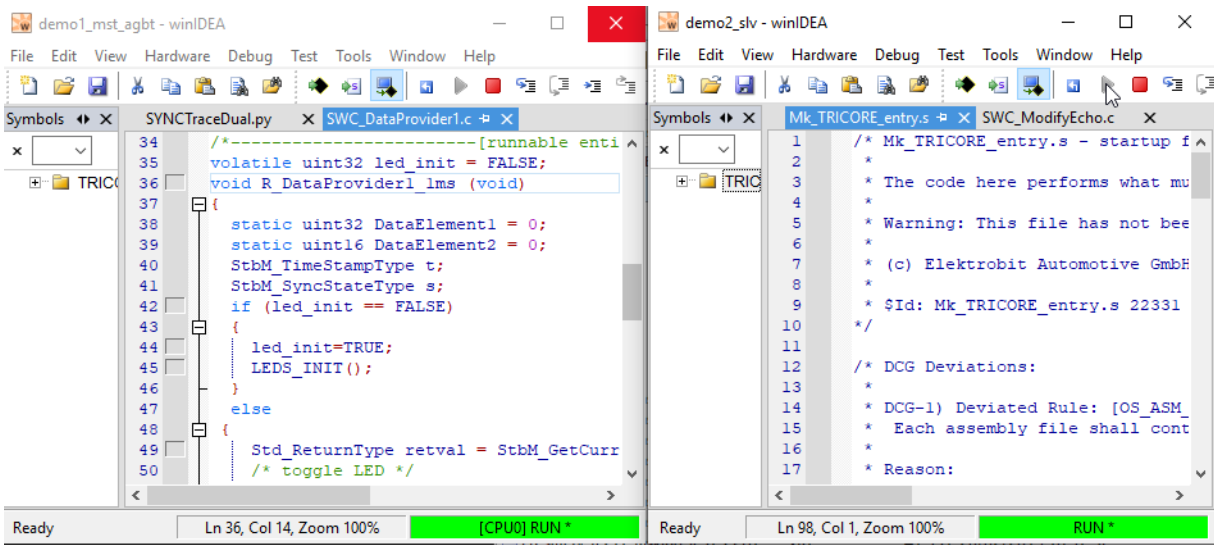
Ready | Ln 36, Col 1, Zoom 100% | [CPU0] STOP *

```
/* Mk_TRICORE_entry.s - startup f
 *
 * The code here performs what mu
 *
 * Warning: This file has not bee
 *
 * (c) Elektrobit Automotive GmbH
 *
 * $Id: Mk_TRICORE_entry.s 22331
*/

/* DCG Deviations:
 *
 * DCG-1) Deviated Rule: [OS_ASM_
 *  Each assembly file shall cont
 *
 * Reason:
```

Ready | Ln 98, Col 1, Zoom 100% | SUSPENDED *

Figure 20: A Breakpoint hit on the Master (here on CPU0 of the Master SoC), causes the Slave to enter »SUSPENDED*« State.

Starting the CPU (RUN) on the Slave, also causes the Master to enter »**RUN***« state (see Figure 21).



```
/*-----------------------[runnable enti
volatile uint32 led_init = FALSE;
void R_DataProvider1_1ms (void)
{
  static uint32 DataElement1 = 0;
  static uint16 DataElement2 = 0;
  StbM_TimeStampType t;
  StbM_SyncStateType s;
  if (led_init == FALSE)
  {
    led_init=TRUE;
    LEDS_INIT();
  }
  else
  {
    Std_ReturnType retval = StbM_GetCurr
    /* toggle LED */
```

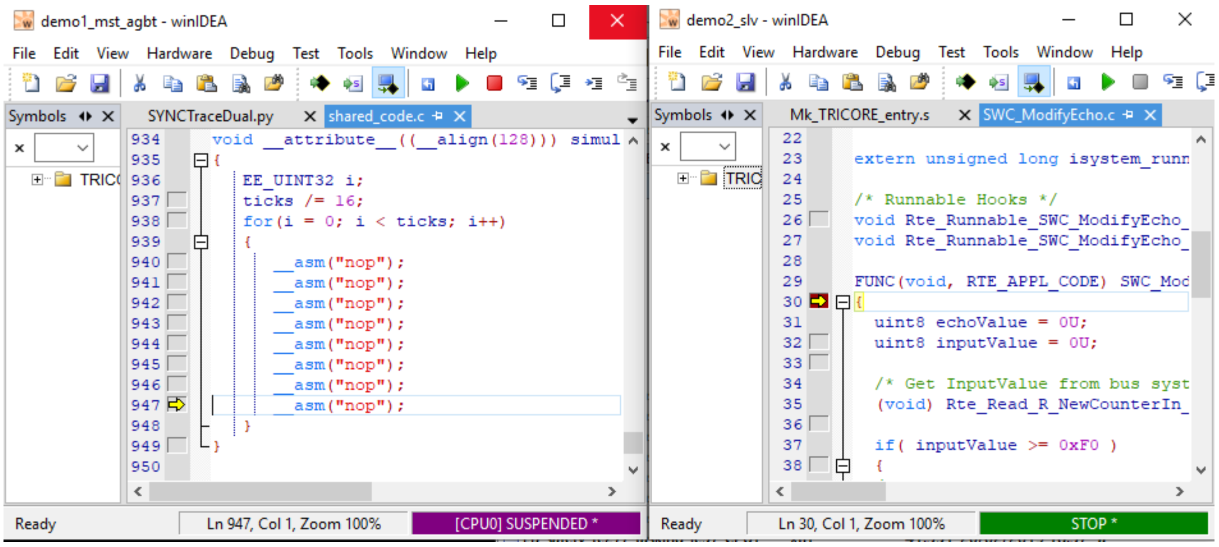
Ready | Ln 36, Col 14, Zoom 100% | [CPU0] RUN *

```
/* Mk_TRICORE_entry.s - startup f
 *
 * The code here performs what mu
 *
 * Warning: This file has not bee
 *
 * (c) Elektrobit Automotive GmbH
 *
 * $Id: Mk_TRICORE_entry.s 22331
*/

/* DCG Deviations:
 *
 * DCG-1) Deviated Rule: [OS_ASM_
 *  Each assembly file shall cont
 *
 * Reason:
```

Ready | Ln 98, Col 1, Zoom 100% | RUN *

Figure 21: A Run Operation on the Slave also causes the Master to enter “RUN*” State.

A Breakpoint hit on the Slave also causes the Master to halt. The CPU enters the AURIX-specfic »**SUSPENDED***« state (see Figure 22).



```
void __attribute__((__align(128))) simul
{
  EE_UINT32 i;
  ticks /= 16;
  for(i = 0; i < ticks; i++)
  {
      __asm("nop");
      __asm("nop");
      __asm("nop");
      __asm("nop");
      __asm("nop");
      __asm("nop");
      __asm("nop");
      __asm("nop");
  }
}
```

Ready | Ln 947, Col 1, Zoom 100% | [CPU0] SUSPENDED *

```
extern unsigned long isystem_runn

/* Runnable Hooks */
void Rte_Runnable_SWC_ModifyEcho_
void Rte_Runnable_SWC_ModifyEcho_

FUNC(void, RTE_APPL_CODE) SWC_Mod
{
  uint8 echoValue = 0U;
  uint8 inputValue = 0U;

  /* Get InputValue from bus syst
  (void) Rte_Read_R_NewCounterIn_

  if( inputValue >= 0xF0 )
  {
```

Ready | Ln 30, Col 1, Zoom 100% | STOP *

Figure 22: A Breakpoint hit on the Slave, causes the Master to enter »SUSPENDED*« State.

3.3 Single-Step Operation

Single-Step operations on the Master do not have any effect on the Slave. The Slave remains in »SUSPENDED*« state.

While in »SUSPENDED* state, single-step operations (source-level or assembly-level) are not possible. Single-Step is only possible when the SoC is in »STOP*« state.

4 Synchronized Trace Operation

Performing a synchronized trace recording requires the following steps:

1. **Alignment of iC5700 BlueBox Time Offset**. This needed for the trace visualization in the Profiler Timeline View. The Time Offset alignment aligns the time offset from time 0 to the first trace recording, in each Profiler Timeline. It has not influence on the overall trace synchronization accuracy.

2. **Start of both Trace Recorders**. The operations starts the generation of trace messages on the target. However, as the recorder (iC5700) is set to »ON Trigger« the iC5700 will not record any trace data until the occurrence of the Trace Trigger.

3. **Trace Trigger**. A trace trigger can be injected either on the Slave or on the Master winIDEA instance. The trigger is propagated via the FNET to both Master and Slave iC5700 trace recorder. The propagation delay of the trace trigger via FNET has no influence on the overall trace synchronization accuracy.

The sample Python script shown in performs the steps described above.

```
import os
import sys
import isystem.connect as ic

mydir = <User-specific Workspace Location>

# Master Workspace location
workspaceLocation1 = os.path.join(mydir,
'Demo1/winidea/Tasks_ISR2_States_NoOnInstr', 'demo1_mst_agbt.xjrf')
# Master TRD file name
trdName1 = 'demo1_mst_agbt.trd'

# Slave Workspace location
workspaceLocation2 = os.path.join(mydir, 'Demo2/winidea', 'demo2_slv.xjrf')
# Slave TRD file name
trdName2 = 'demo2_slv.trd'

# Create all necessary Objects to control the Master.
cmgr_MST = ic.ConnectionMgr()
connectionConfig = ic.CConnectionConfig()
connectionConfig.workspace(workspaceLocation1)
cmgr_MST.connect(connectionConfig)
debug_MST = ic.CDebugFacade(cmgr_MST)
ideCtrl_MST = ic.CIDEController(cmgr_MST)
traceDoc_MST = ic.CTraceController(cmgr_MST, trdName1, 'w')

# Create all necessary Objects to control the Slave.
cmgr_SLV = ic.ConnectionMgr()
connectionConfig = ic.CConnectionConfig()
connectionConfig.workspace(workspaceLocation2)
cmgr_SLV.connect(connectionConfig)
ideCtrl_SLV = ic.CIDEController(cmgr_SLV)
traceDoc_SLV = ic.CTraceController(cmgr_SLV, trdName2, 'w')


# Adjust BlueBox (iC5700) Time Offset (just needed for aligned display in
Profiler Timeline).
```

```
outParams = ic.StrStrMap()
inParams  = ic.StrStrMap()
ideCtrl_MST.serviceCall("/IOPEN/HW.HW.GetBBTime", inParams, outParams)
timeValue = outParams['Time']

# Adjust Time Offset of Master BlueBox.
ideCtrl_MST.setOption_str('/Document/' + trdName1 +
'/Loader.HW.Data.TimerOffsetActivation', timeValue)

# Adjust Time Offset of Master BlueBox.
ideCtrl_SLV.setOption_str('/Document/' + trdName2 +
'/Loader.HW.Data.TimerOffsetActivation', timeValue)


# Start Trace Recorder on Master and Slave. Both wait for a common Trace
Trigger.
traceDoc_MST.start()
traceDoc_SLV.start()

# Start Master CPU. This also start Slave CPU due to sync debug setup.
debug_MST.run()

# Inject SYNC Trace start on Trigger Channel 1 (i.e. TraceTrig).
ideCtrl_MST.serviceCall("/IOPEN/HW.FNet.FTrig_Inject", "TC:1")

# Done. Disconnect from both Workspaces again.
cmgr_SLV.disconnect()
cmgr_MST.disconnect()
```

Listing 1: Sample Python Script to start a synchronized Trace Operation

As stated above, the trace trigger is propagated throught the entire system via FNET.
In addition to the trace trigger, also a trace clock is propagated to all trace recorders.
Synchronzed debug start/stop operations are as well controlled via FNET.

A simplified Block Diagram of the Overall iSYSTEM FNET Concept is depicted in Figure 23.

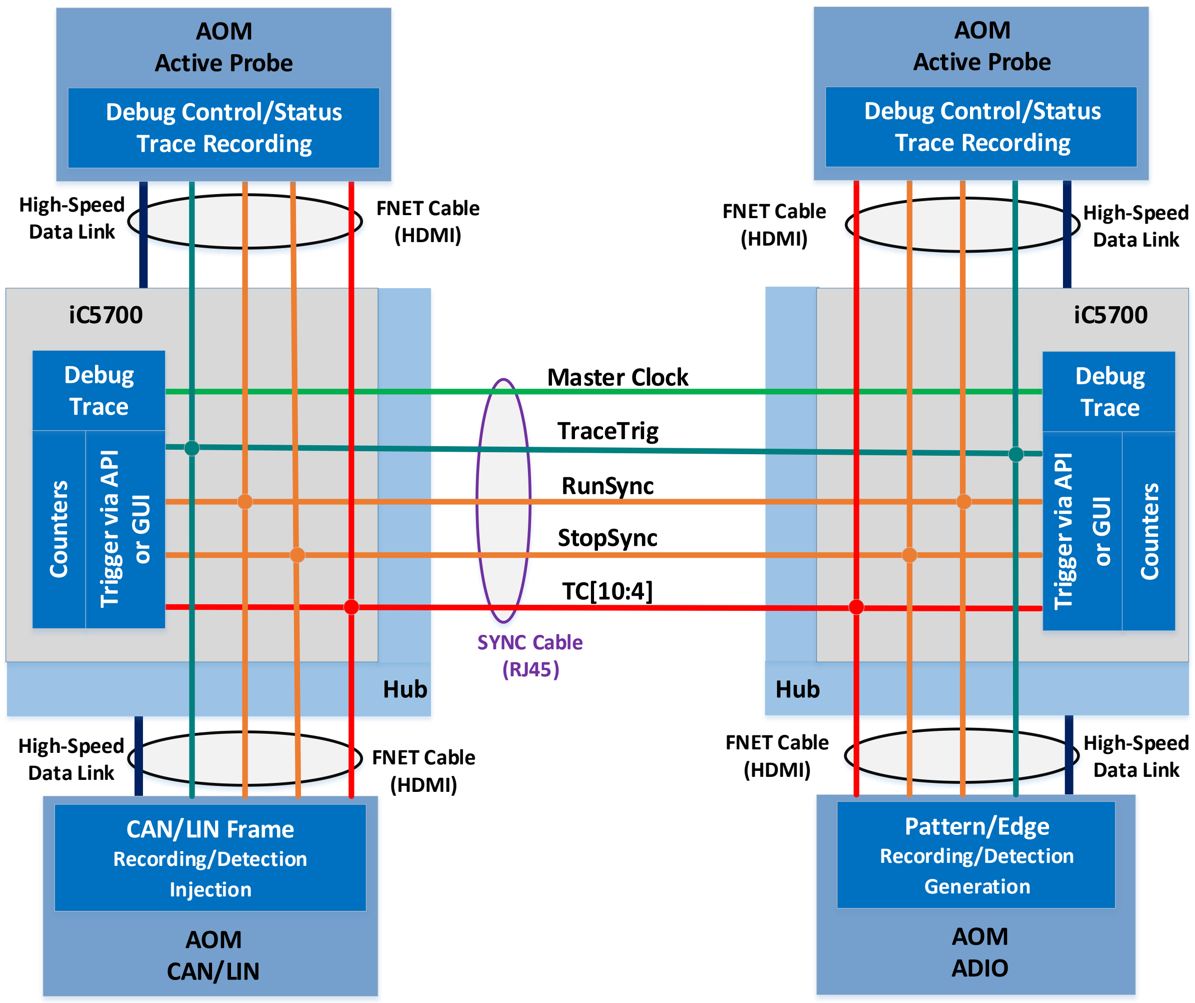


Figure 23: Simplified Block Diagram of the Overall iSYSTEM FNET Concept

5 Profiler Timeline View

Figure 24 shows a sample Profiler Timeline view of a synchronized dual-AURIX trace recording. Besides the trace of the AUTOSAR OS tasks and ISR2s, also the CAN frame are recorded on both Socs (also synchronized to AURIX MCDS trace).

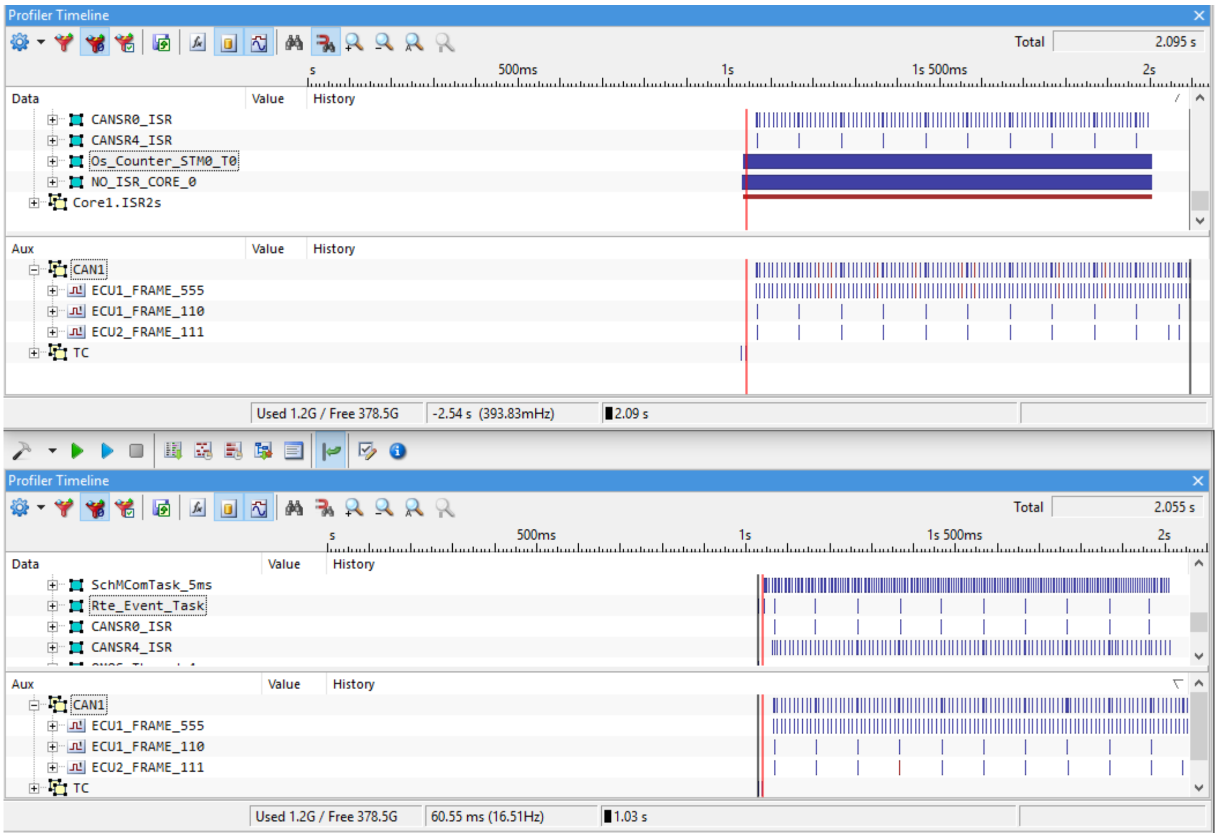


Figure 24: Sample Profiler Timeline View of a Synchronized Trace on two AURIX Devices
(top = Master, bottom = Slave)

Figure 25 zoomes into a specific section of the Profiler Timeline view of Figure 24.
The Profiler Timeline on the top shows the OS ISR2 and CAN bus activities on the Master.
The Profiler Timeline on the bottom shows the OS thread (task or ISR2) and CAN bus activities on the Slave.
The scenario shown here is the following:

1. The Masters sents out a CAN frame with ID 0x110. Transmit competion causes a TX ISR2, CANSR0_ISR (blue cursor).

2. The Slave receives the CAN frame 0x110, issues an RX interrupt, CANSR4_ISR, which spawns an OS thread (i.e. task), Rte_Event_Task. A Runnable within this task (not shown here) responds by sending out a CAN frame with ID 0x111. Transmit competion causes a TX ISR2, CANSR0_ISR (yellow cursor).

3. The Master receives this CAN frame 0x111 and issues an RX interrupt, CANSR4_ISR (yellow cursor).

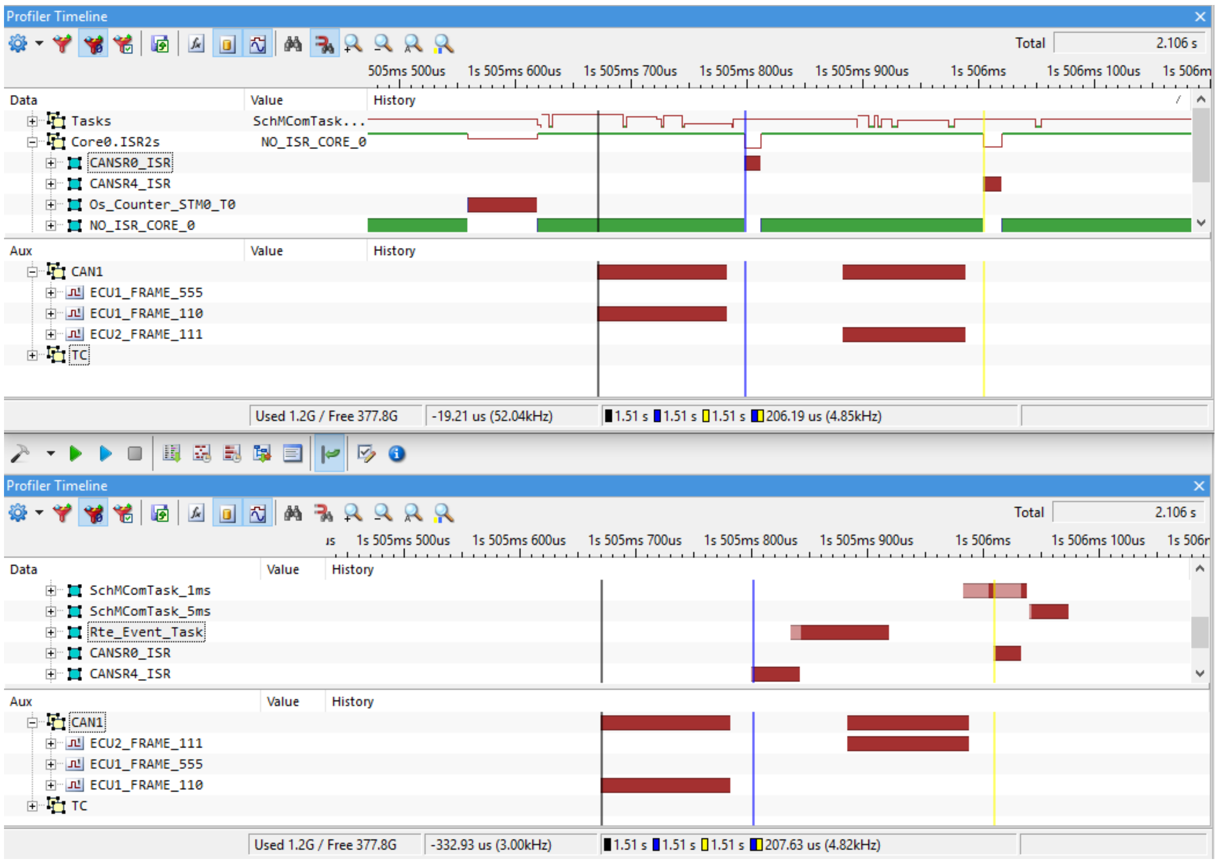


Figure 25: Zoom-In of the Sample Profiler Timeline View of Figure 24.

6 Technical support

6.1 Online resources

Online Help ▸ winIDEA and testIDEA online help	Knowledge Base ▸ Tips & tricks categorized by issue type and architecture	Tutorials ▸ From a beginner to an expert
Technical Notes ▸ How-tos for winIDEA functionalities with scripts	Application Notes ▸ How-to notes on advanced use-cases	Webinars ▸ Technical webinars about ISYSTEM tools with use cases

6.2 Contact

Please visit https://www.isystem.com/contact.html for contact details.