

Introduction

This document provides description of Lauterbach tools to connect and debug devices of the SPC56x families that support multicore.

Supported devices list:

- SPC56EL60xx
- SPC56EL70xx
- SPC56AP60xx
- SPC56EC7xx

Contents

1	Dual-core architecture	4
2	ST dual-core products	5
3	Setting up Lauterbach environment	6
3.1	Dual-core debug environment	6
3.2	Basic configuration	7
3.3	Lauterbach Batchfile	10
3.3.1	System synchronization reset	11
3.3.2	Setting debugging path	12
3.3.3	Configuring PowerView for multicore debugging	13
3.3.4	Attach to slave core	13
3.3.5	MMU Initialization	14
3.3.6	SRAM initialization	14
3.3.7	Flash declaration	15
3.3.8	Loading data into Flash	16
3.3.9	Setting up cores synchronization	17
4	Revision history	18

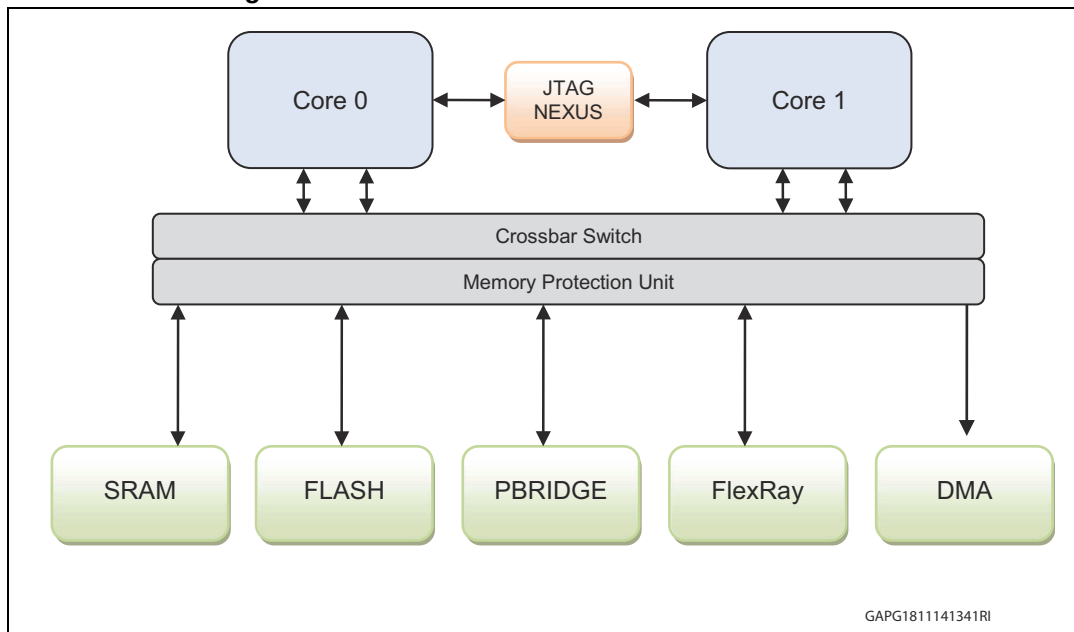
List of figures

Figure 1.	Power Architecture dual-core architecture	4
Figure 2.	Lauterbach Power Architecture multicore license.	6
Figure 3.	Lauterbach Power Architecture multicore debug schema	7
Figure 4.	T32Start dual-core example	8
Figure 5.	T32Start dual-core – core1 example	9
Figure 6.	T32Start dual-core – core2 example	10
Figure 7.	Batchfile - intercom & synch reset	11
Figure 8.	Batchfile – setting debugger working directory and BDM clock	12
Figure 9.	Batchfile – setup for multi-core debugging	13
Figure 10.	Batchfile – Slave core attach	13
Figure 11.	Batchfile – MMU initialization	14
Figure 12.	Batchfile – SRAM initialization	14
Figure 13.	Batchfile – FLASH declaration	15
Figure 14.	Batchfile - loading data	16
Figure 15.	Batchfile - loading only source	16
Figure 16.	Synchronization between cores	17

1 Dual-core architecture

Figure 1 is an example for Power Architecture® dual-core architecture.

Figure 1. Power Architecture dual-core architecture



2 ST dual-core products

Table 1. ST's Dual-core microcontrollers

Device	Cores	Operating modes	Class
SPC56EL60	z4 + z4	LSM + DPM	Chassis & safety
SPC56EL70	z4 + z4	LSM + DPM	Chassis & safety
SPC56AP60	z0 + z0	DPM	Chassis & safety
SPC56EC7	z4 + z0	DPM	Body

3 Setting up Lauterbach environment

The presented application example is for dual core microcontroller in Decoupled mode. Demonstration is done for ST's SPC56EL70 processor from chassis-safety family with 128kB RAM and 2MB Flash. Tools used are Lauterbach debugger and TRACE32 debugging interface. SPC56x families device combines DPM (decoupled) and LSM (lock-step) modes. There are many ways how to debug Multicore processor. This section is about Lauterbach debugger used for Multicore debugging, in our case dual-core debugging.

3.1 Dual-core debug environment

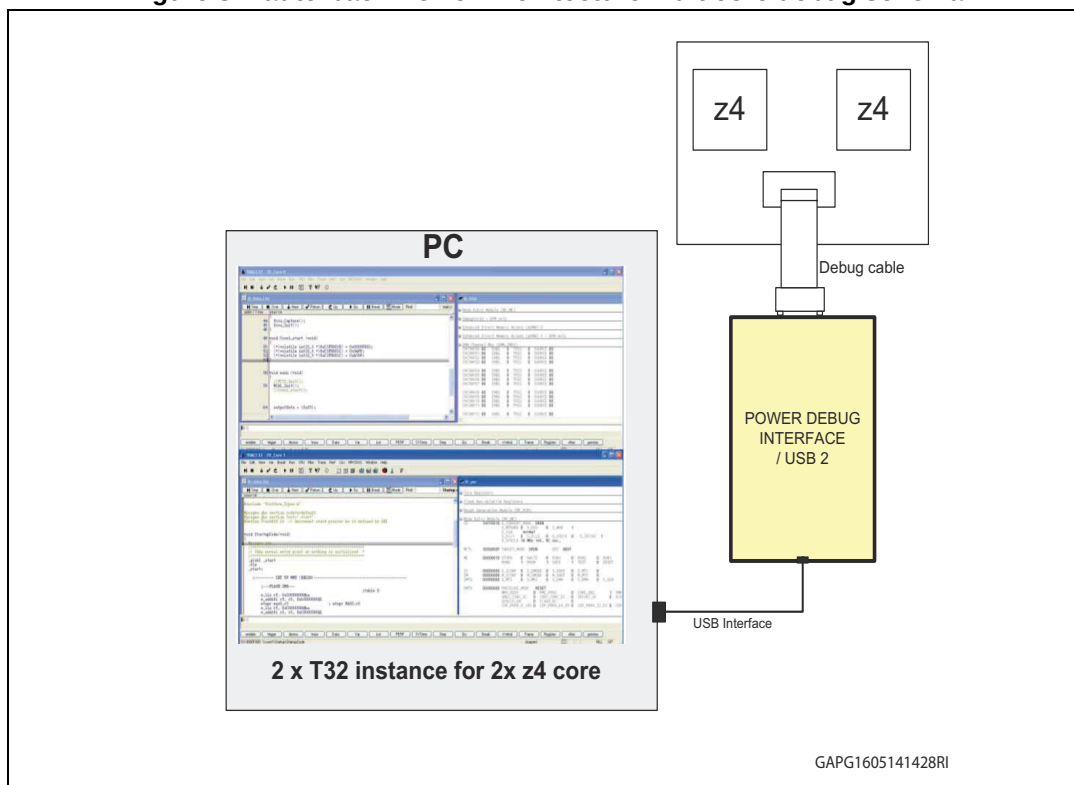
There is a possibility to debug dual-core processor with single TRACE32 PowerView window, but for better orientation and easier debugging there is the possibility to start multiple PowerView windows. The term **Single Device Solution** is used if one Lauterbach device is used to debug all cores. Precondition is that the debug cable contains licenses for all cores that should be debugged.

Figure 2. Lauterbach Power Architecture multicore license



The following example describes how to start two TRACE32 instances. On a multicore chip both cores are debugged via a joint JTAG interface by using a POWER DEBUG INTERFACE / USB 2.

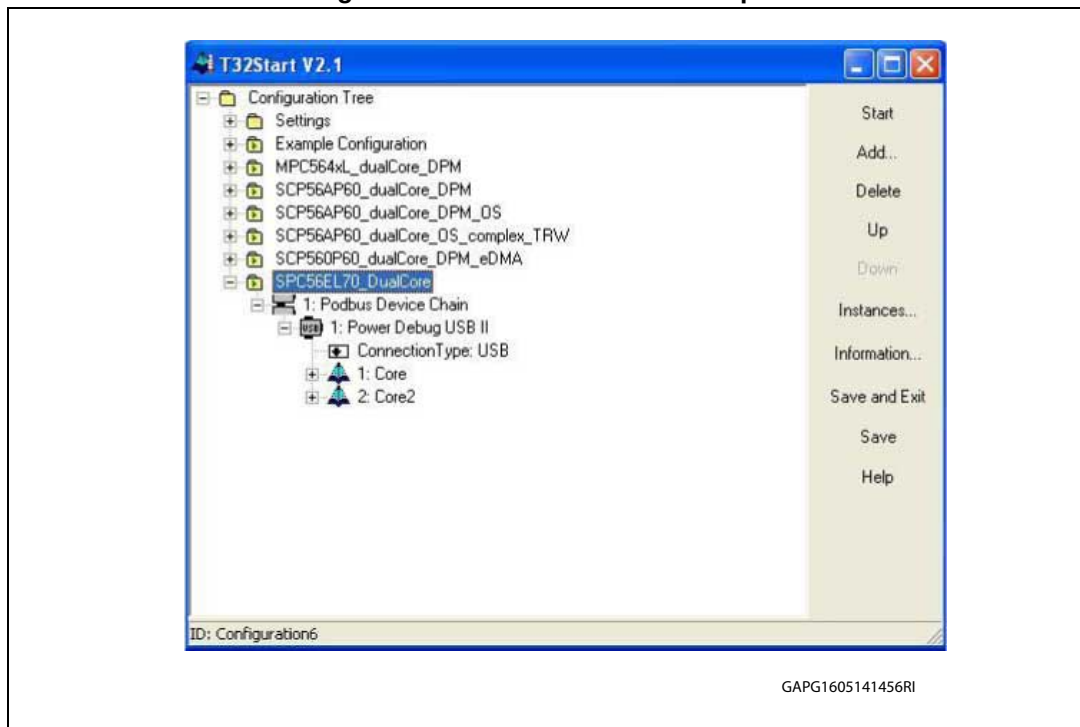
Figure 3. Lauterbach Power Architecture multicore debug schema



3.2 Basic configuration

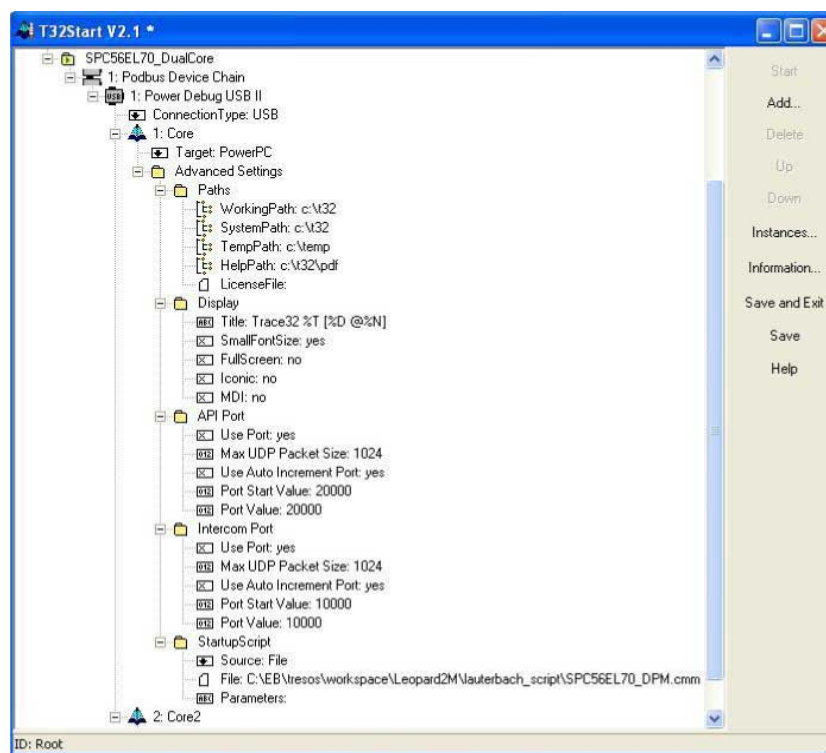
To work with 2 TRACE32 instances it is necessary preparing TRACE32 accordingly the requirements. T32Start instance is a GUI designed exactly for this purpose. Let's create a new configuration by setting cursor on Configuration tree and click right mouse adding a configuration. Keep on in this way and setup connection for 2 cores like in [Figure 4](#).

Figure 4. T32Start dual-core example



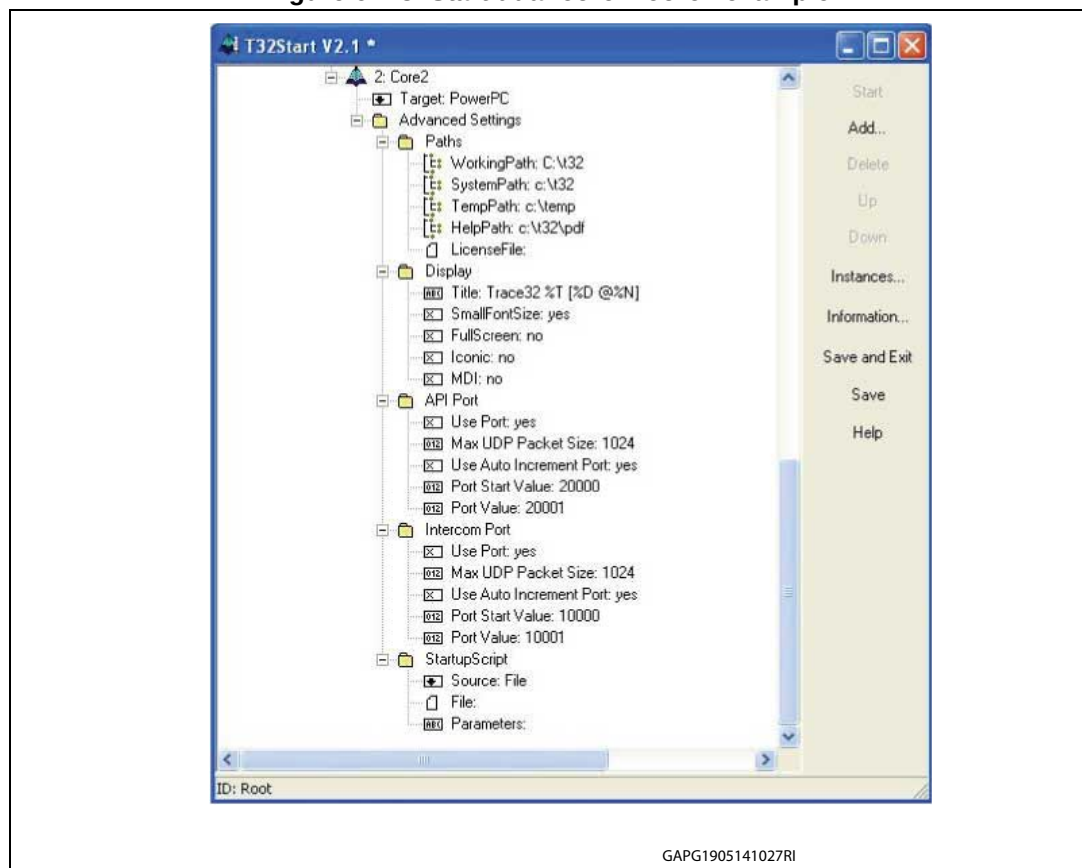
In [Figure 5](#) you can find correct settings for first (master) core followed by [Figure 6](#) illustrating second (slave) core settings. As you can see it is used only 1 Lauterbach batchfile for both cores.

Figure 5. T32Start dual-core – core1 example



GAPG1605141501RI

Figure 6. T32Start dual-core – core2 example



In the API port section the field “Use Port” can be set to “no” if no remote control is necessary for TRACE32. (This applies both to [Figure 5](#) and [Figure 6](#)).

It is possible starting when everything is prepared in T32Start GUI for dual-core debug. It is necessary set the cursor on Configuration (Top level of our project) and press start, two TRACE32 instances are opened.

In the configuration select specific batch file that complete the initialization after the two instances of TRACE32 are loaded.

If any batch file is selected, there is not connection to the cores and any specific initialization that can be done after the set up in each instance.

3.3 Lauterbach Batchfile

It needs several commands to Initialize and set the debug environment and target. They can be combined in a batch file, with the default extension “cmm”. TRACE32 has its own command language PRACTICE to execute batch jobs when loaded. All commands of the TRACE32 development tools, commands for program flow, conditional commands and I/O commands are allowed. Also debugging of a PRACTICE program is supported. Let's go step by step through prepared batch file for dual-core processor SPC564EL70.

3.3.1 System synchronization reset

The 'InterCom' system allows the exchange of data between different TRACE32 systems. The exchange can be based on TCP/IP or, if not possible, through files on a network drive. The destination system is defined by an InterCom name. This name is either the name and port number of a UDP port used by this TRACE32 system or a file name. SYNch.RESet command resets configuration of synchronization and synchronization is disabled.

Figure 7. Batchfile - intercom & synch reset

```
15 ;start core 1 debugger software and check intercom
16
17 &core0=""
18 &core1="intercom localhost:10001"
19
20 ;wait some time until the software finished booting
21 ;In case of error "no response from InterCom" wait for startup of debugger 2
22 ;delay command = "intercom.wait localhost:10001" or " wait 3.s"
23
24 intercom.wait localhost:10001
25
26 ;=====
27 ;name TRACE32 windows
28 &core0 Title "TRACE32 - ZO_Core 0"
29 &core1 Title "TRACE32 - ZO_Core 1"
30
31 ;reset both cores
32 &core0 SYStem.RESet
33 &core1 SYStem.RESet
34
35 ;reset synchronization
36 &core0 SYNCH.RESet
37 &core1 SYNCH.RESet
38
39 ;=====
```

GAPG1905141036RI

3.3.2 Setting debugging path

Figure 8. Batchfile – setting debugger working directory and BDM clock

```

39 ;=====
40 ;arrange debugger windows
41 &core0 framepos -183. 0. 181. 26.
42 &core1 framepos -183. 39.3 181. 26.
43
44 ;=====
45 ;print message to distinguish between cores
46 &core0 print "**** hello core 0 ****"
47 &core1 print "**** hello core 1 ****"
48
49 ;=====
50 ;Set path of core 1 debugger to the path of core 0 debugger
51 &path=OS.PWD()
52 &core1 cd &path
53
54 ;=====
55 ;System settings
56 &core0 SYStem.BdmClock 4.MHz
57 &core1 SYStem.BdmClock 4.MHz
58
59 ;detect used processor
60 SYStem.CPU SPC56EL70
61 SYStem.DETECT CPU
62 &cpu=CPU()
63
64 ;=====

```

GAPG1905141045RI

In the line 50 of the described script it is reported the command to store the working directory into "path" and change the path for working directory of core1 to the path of core0 debugger.

OS.PWD() - returns the name of the working directory as a string.

Select the frequency for the debug interface. For multicore debugging, it is recommended to set the same JTAG frequency for all cores.

In the line 56 the BDM clock frequency is selected.

SYStem.CPU SPC56XX - sets up selected microcontroller. This is mainly used when there is not a support for new derivative of microcontroller, and user wants set by hand processor from same family.

SYStem.DETECT CPU - detects automatically connected microcontroller if it is in list of supported microprocessors.

CPU() - returns the name of the processor as string (same as **STATE.PROCESSOR**). It's the same as selected with **SYStem.CPU**

3.3.3 Configuring PowerView for multicore debugging

Figure 9. Batchfile – setup for multi-core debugging

```

65 ;Configure both PowerView instances for detected CPU
66
67 &core0 SYStem.CPU &cpu
68 &core1 SYStem.CPU &cpu
69
70 &core0 SYStem.Option.WATCHDOG OFF
71 &core1 SYStem.Option.WATCHDOG OFF
72
73 ;=====
74 ;setup for multi-core debugging
75
76 &core0 SYStem.Config.Core 1.
77 &core1 SYStem.Config.Core 2.
78 &core1 SYStem.MultiCore.slave ON
79
80 ;core 0: halt on POR
81 &core0 SYStem.Up
82

```

GAPG1905141229RI

Setting up **System.CPU &cpu** for both cores setup the detected processor for both TRACE32 PowerView instances.

System.Option.WATCHDOG OFF - set watchdog off in system settings.

&core0 System.Config.Core 1. - Commonly one TRACE32 instance is used to debug one core (core view). If the target provides a joint debug interface for several cores it is necessary to inform the TRACE32 instance which core it controls for debugging. The command **SYSTEM.CONFIG.CORE** allows to specify the core within the chip that is controlled by each TRACE32 instance.

SYStem.CONFIG Slave ON - If more than one debugger serves the debug interface of the core, only one debugger must be allowed to reset the core. Therefore one debugger is designated as master, the other debuggers as slaves.

&core0 System.Up - This command resets the CPU (HRESET), enters the debug mode and stops the CPU at the reset vector.

3.3.4 Attach to slave core

Figure 10. Batchfile – Slave core attach

```

83 ;=====
84 ;core 1: still in reset, attach only
85 &core1 SYStem.Mode.Attach
86
87 ;core 1: on a Break while in reset, the debugger will stop core 1 as soon as it comes out of reset
88 &core1 Break
89
90 ;=====

```

GAPG1905141232RI

System.Mode.Attach – This command works similar to the **Up** command. The difference is that the target CPU is not reset. The BDM/JTAG/COP interface is synchronized and the

CPU state are read out. After this command the CPU is in the **SYStem.Up** mode and can be stopped for debugging.

&core1 Break – Breaks the core1 (slave). Core1 is still held in reset, the debugger stops core1 as soon as it comes out of reset.

3.3.5 MMU Initialization

Figure 11. Batchfile – MMU initialization

```

77 ; setup MMU for flash, RAM and register access
78 MMU.Set TLB1 0x0 0x00000000 0x00000000 0x00000000
79 MMU.Set TLB1 0x1 0xC0000500 0xFFFF000A 0xFFFF003F
80 MMU.Set TLB1 0x2 0xC0000700 0x20000000 0x2000003F
81 MMU.Set TLB1 0x3 0xC0000400 0x40000008 0x4000003F
82 MMU.Set TLB1 0x4 0xC0000500 0xC3F00008 0xC3F0003F
83 MMU.Set TLB1 0x5 0xC0000700 0x00000000 0x0000003F

```

GAPG1905141252RI

The correct MMU (memory management unit) initialization is essential for processors which have MMU. Without doing so user would not be able to access uninitialized memory locations. Refer to the reference manual for correct MMU settings. MMU initialization must not be forgotten. Some debuggers can perform MMU initialization automatically from script, but after reset of target processor the MMU table is lost and have to be initialized again from user startup.

MMU.TLB1.SET – sets the MAS registers in [Figure 11](#).

3.3.6 SRAM initialization

Don't forget that dual-core processors usually have 2 SRAM modules. For the second core SRAM can be initialized later when it is started from second core startup.

Figure 12. Batchfile – SRAM initialization

```

100 ;=====
101 ;initialize internal SRAM -> DPM is set
102
103 ;initialize internal SRAM (64KB)
104 Data.Set EA:0x40000000--0x4000ffff %Quad 0x0
105
106 ;initialize internal SRAM (64KB)
107 Data.Set EA:0x50000000--0x5000ffff %Quad 0x0
108
109 ;=====

```

GAPG2005141038RI

Data.Set – This command fills selected memory with defined values. In our case it fill SRAM memory content with 0x0. After executing of this command SRAM content in selected range is 0x0.

3.3.7 Flash declaration

Figure 13. Batchfile – FLASH declaration

```

109 ;=====
110 ; Flash declaration
111
112 FLASH.RESet
113
114 ; Low address space 256k
115 FLASH.Create 1. ($flashbase+0x00000000)+0x03fff TARGET Quad 0. ; L0 (16kB)
116 FLASH.Create 1. ($flashbase+0x00004000)+0x03fff TARGET Quad 1. ; L1 (16kB)
117 FLASH.Create 1. ($flashbase+0x00008000)+0x03fff TARGET Quad 2. ; L2 (16kB)
118 FLASH.Create 1. ($flashbase+0x0000c000)+0x03fff TARGET Quad 3. ; L3 (16kB)
119 FLASH.Create 1. ($flashbase+0x00010000)+0x03fff TARGET Quad 4. ; L4 (16kB)
120 FLASH.Create 1. ($flashbase+0x00014000)+0x03fff TARGET Quad 5. ; L5 (16kB)
121 FLASH.Create 1. ($flashbase+0x00018000)+0x03fff TARGET Quad 6. ; L6 (16kB)
122 FLASH.Create 1. ($flashbase+0x0001c000)+0x03fff TARGET Quad 7. ; L7 (16kB)
123 FLASH.Create 1. ($flashbase+0x00020000)+0x0ffff TARGET Quad 8. ; L8 (64kB)
124 FLASH.Create 1. ($flashbase+0x00030000)+0x0ffff TARGET Quad 9. ; L9 (64kB)
125
126 ; Mid address space 256k
127 FLASH.Create 2. ($flashbase+0x00040000)+0x1ffff TARGET Quad 0. ; M0 (128kB)
128 FLASH.Create 2. ($flashbase+0x00060000)+0x1ffff TARGET Quad 1. ; M1 (128kB)
129
130 ; High address space 1,5M
131 FLASH.Create 3. ($flashbase+0x00080000)+0x3ffff TARGET Quad 0. ; H0 (256kB)
132 FLASH.Create 3. ($flashbase+0x000c0000)+0x3ffff TARGET Quad 1. ; H1 (256kB)
133 FLASH.Create 3. ($flashbase+0x00100000)+0x3ffff TARGET Quad 2. ; H2 (256kB)
134 FLASH.Create 3. ($flashbase+0x00140000)+0x3ffff TARGET Quad 3. ; H3 (256kB)
135 FLASH.Create 3. ($flashbase+0x00180000)+0x3ffff TARGET Quad 4. ; H4 (256kB)
136 FLASH.Create 3. ($flashbase+0x001c0000)+0x3ffff TARGET Quad 5. ; H5 (256kB)
137
138 ;=====
139 ; Shadow row
140 FLASH.Create 4. ($shadowbase+0x00000000)+0x3fff NOP Quad ;(16kB)
141
142 FLASH.TARGET E:&rambase E:&rambase+0x2000 0x1000 C:\T32\demo\powerpc\flash\quad\c90f1564x1.bin /STACKSIZE 0x0200
143
144 ;=====

```

GAPG19051411310RI

SPC56EL70 has three main blocks of flash – low, middle, high which are divided into partitions. Correct initialization is essential for work with flash memory. Follow reference manual of microcontroller and set flash blocks accordingly. In case flash is not declared correctly usually flash erase or change of flash content is not successful and TRACE32 returns a memory access error. If the FLASH has sectors of different size (boot block devices) one **FLASH.Create** command has to be entered for each sector size.

Special case is shadow and test flash. Usually user is not changing content of those flashes and they are set as NOP.

The family_code TARGET must be selected in the command FLASH.Create, if target controlled FLASH programming is used. The definitions for target based FLASH programming are done via the FLASH.TARGET command. In ST processors the code and data area for the flash algorithm must be allocated from internal processor memory.

FLASH.TARGET E:&rambase – is used to inform TRACE32 about an appropriate RAM location where the flash programming algorithm can be downloaded and where memory is available for the flash programming data and for the flash algorithm data.

3.3.8 Loading data into Flash

Figure 14. Batchfile - loading data

```

147 FLASH.ReProgram ALL /Erase
148
149 ;load sources+code for core1 (SLAVE)
150 &core1 Data.LOAD C:\work\SPC56EL70_DPM_example\ghs\core1.elf E:0x00--(&flashsize-1)
151 ;load sources+code for core0 (MASTER)
152 &core0 Data.LOAD C:\work\SPC56EL70_DPM_example\ghs\core0.elf E:0x00--(&flashsize-1)
153
154 FLASH.ReProgram off
155
156 ;=====

```

GAPG2005141043RI

Loading prepared code to target processor is done via **&coreX Data.LOAD**. There is a recommended procedure to load code for slave cores first and then load master core code. With command **FLASH.ReProgram ALL** is user allowing programming of flash. Optional is using erase of flash. In our case whole flash is erased before any operation is done.

Lauterbach debugger reads the flash content modifying it, but only inside debugger. To close flash algorithm command **FLASH. ReProgram Off** is used and then Lauterbach flash its modified content into microprocessor. If any error occurs with closing flash algorithm user is notified in TRACE32 PowerView when “OFF” command is executed. This basically means no real change of flash content until flash routine is finished.

Figure 15. Batchfile - loading only source

```

;program flash
FLASH.ReProgram ALL /Erase
Data.Load core0.elf E:0x00--(&flashsize-1) /NoYmbol
Data.Load core1.elf E:0x00--(&flashsize-1) /NoYmbol

FLASH.ReProgram off

;load debug symbols
&core0 Data.Load core0.elf /NoCODE
&core1 Data.Load core1.elf /NoCODE

;reset processor to take over new RCHW from flash
&core0 SYStem.Up

```

GAPG2005141045RI

The /NoCODE option of Data.LOAD is used to suppress the code download. Only symbolic information is loaded in TRACE32 and no flash programming occurs.

3.3.9 Setting up cores synchronization

Figure 16. Synchronization between cores

```

163 ;=====
164 ;setup for sync step/go/break
165 &core0 SYNCH.CONNECT localhost:10001
166 &core0 SYNCH.MasterGO ON
167 &core0 SYNCH.MasterBREAK ON
168 &core0 SYNCH.MasterSTEP ON
169 &core0 SYNCH.SlaveGO ON
170 &core0 SYNCH.SlaveBREAK ON
171 &core0 SYNCH.SlaveSTEP ON
172
173 &core1 SYNCH.CONNECT localhost:10000
174 &core1 SYNCH.MasterGO ON
175 &core1 SYNCH.MasterBREAK ON
176 &core1 SYNCH.MasterSTEP ON
177 &core1 SYNCH.SlaveGO ON
178 &core1 SYNCH.SlaveBREAK ON
179 &core1 SYNCH.SlaveSTEP ON
180
181 ;switch to HLL (High level language) mode debugging, if cannot be displayed asm code is shown
182 &core0 MODE.HLL
183 &core1 MODE.HLL
184
185 &core0 data.list
186 &core1 data.list
187

```

GAPG2005141101RI

The SYnch command is used for Start/Stop synchronization in a multi-processor or multi-core environment.

SYnch.CONNECT - Establishes a connection to another emulator by using the InterCom system. Multiple names can be used to connect to multiple systems.

SYnch.MasterGo – Starting this emulator all other emulators start, the **SlaveGo** function is activated.

SYnch.MasterBreak – Breaking this emulator system simultaneously stop all emulators which have the **SlaveBreak** option activated.

SYnch.MasterStep – Stepping this emulator step all other emulators which have the **SlaveStep** function activated.

SYnch.SlaveGo – The emulator starts, when the master emulator starts.

SYnch.SlaveBreak – The emulator stops, when the master emulator stops.

SYnch.SlaveStep – The emulator steps, when the master emulator steps.

To test the INTERCOM system the **INTERCOM.PING** command can be used. If the connection is not working (step/break/go in one window are not made same in second core window even if synch options are all allowed) you can use the ping command. If it returns around 100µs this means you have connected core to itself. Check the Synch.CONNECT settings.

Writing Synch command in TRACE32 PowerView command line brings up windows for synchronization options.

4 Revision history

Table 2. Document revision history

Date	Revision	Changes
19-Nov-2014	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2014 STMicroelectronics – All rights reserved

