# AN11609

## LPC5410x Dual Core Usage

Rev. 1.0 — 4 November 2014 **Application Note**

**Document information**

| Info | Content |
|---|---|
| **Keywords** | LPC5410x, LPC54102, Microcontroller, Dual Core, Cortex-M4F, Cortex-M0+ |
| **Abstract** | Application related information for the Cortex-M4F/M0+ dual core implementation in LPC54102 microcontroller. |

## Revision history

| Rev | Date | Description |
|---|---|---|
| 1.0 | 4 Nov 2014 | First release |

# Contact information

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

# 1. Introduction

The LPC54102 is a member of the LPC54100 family of devices, integrating two 32-bit ARM cores. Targeted for low power applications, this device is optimized for the lowest power consumption in active mode and also provides excellent power numbers in stand-by and power-down modes.

The two cores, a Cortex-M4F and a Cortex-M0+, can work independent of each other or team up to work on shared tasks.

In addition to the information available in the LPC54102 Data Sheet and User Manual, this application note provides suggestions on how to use these two cores and guidelines on how to set up software projects and debug sessions.

The accompanying code example is based on NXP's LPCOpen platform using the free LPCXpresso v7.5.0 IDE and Keil µVision v5.12 with LPC54xxx device family pack (DFP).

http://www.lpcware.com/lpcxpresso/download

http://www.lpcware.com/content/nxpfile/lpcopen-platform

https://www.keil.com/demo/eval/arm.htm

For the software examples the LPCXpresso LPC54102 board is used (OM13077 evaluation kit).

# 2. LPC54102 Dual Core Implementation

## 2.1 Overview

The Cortex-M4F in LPC54102 has the following features:

- MPU and single precision FPU.
- Three interrupt priority levels and VTOR register.
- SysTick timer.
- Sleep mode power saving + NXP's extended modes.
- SW-DP with 8 breakpoints, 4 data watchpoints.

The Cortex-M0+ in LPC54102 has the following features:

- Multiply support in hardware (32-clock cycle version).
- SysTick timer.
- VTOR register.
- Sleep mode power saving + NXP's extended modes.
- SW-DP with 4 breakpoints and 2 data watchpoints.

## 2.2 Dual core system integration

Both ARM cores are implemented as AHB masters (Advanced High Performance Bus), this means that they both have full control of the available resources on the LPC54102 device.

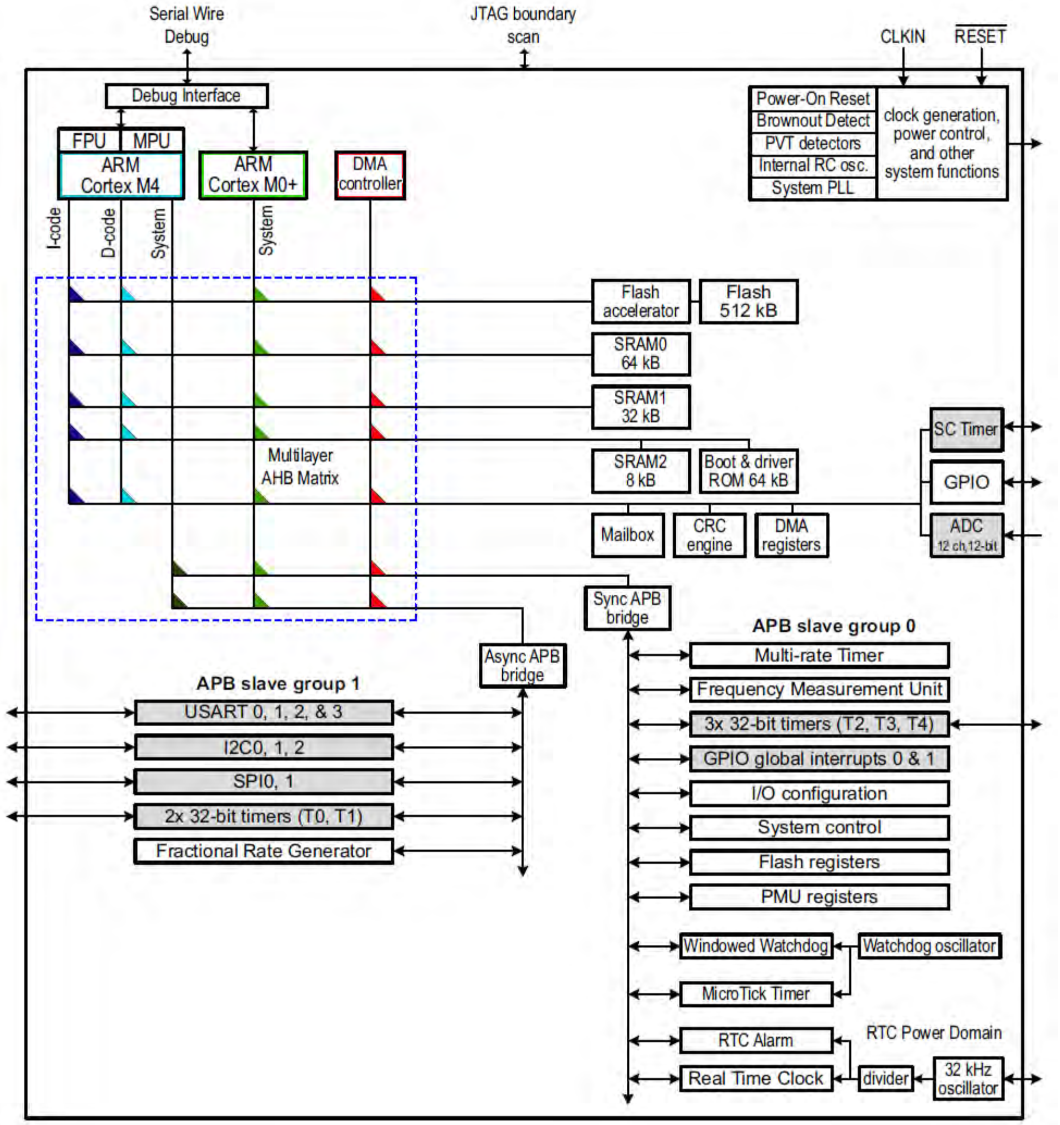


**Fig 1. LPC54102 AHB multilayer matrix**

It is clear that an access to the same resource may result in bus contention (see Fig 1), therefore, proper planning of the tasks for the two cores is beneficial. The part provides adequate flexibility to alleviate this issue.

- Depending on the application setup, one core is designated the master and owns the Flash while the other core uses SRAM for code.

- Access to the flash can happen in parallel, the Cortex-M4F uses the I-code and the D-code bus for code execution whereas the Cortex-M0+ uses the System bus. Access to SRAM can also be split onto SRAM0 and SRAM1.
- Access to the same resource causes bus arbitration.

The resource planning is an important part of a dual core software project because only the Cortex-M4F owns a Memory Protection Unit (MPU).This MPU can be used to prevent the Cortex-M4F to run into areas, such as the stack area of the Cortex-M0+.

On the Cortex-M0+ side this is not possible, there is no safe way to prevent the Cortex-M0+ from poaching in memory areas which are dedicated to the Cortex-M4F.

The two cores are an asymmetric implementation, compared to application processors with two Cortex-A9 for example. So any type of parallel processing would be difficult to implement, due to the different capabilities of the cores and the way the cores share the on-chip resources.

# 3. Core-to-Core Communication

To share tasks between the two cores, a certain level of communication and hand-shaking is required.

The Mailbox unit provides low level features for an inter-core communication. To enable this block the bit MAILBOX in register AHBCLKCTRL0 needs to be set (SYSCON unit).

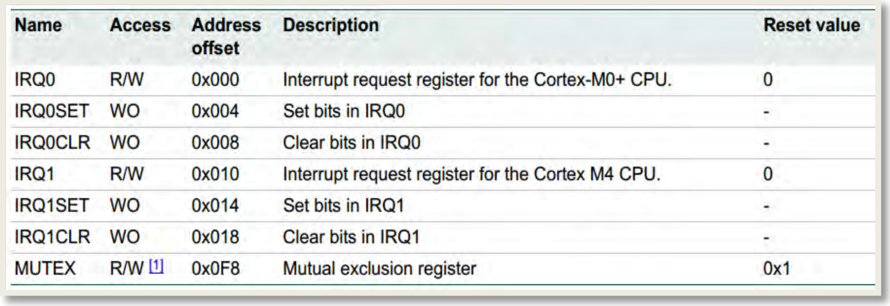
## 3.1 Mailbox implementation

There is no optimal hardware mailbox structure implemented in LPC54102. There is no dedicated hardware controlled FIFO or ring buffer to stream data from one core to the other. The following three mechanisms allow for creation of a mailbox structure in software:

- Core-to-core interrupts
- Mutex register
- Equal access to SRAM memory areas

Compared to the dual core implementation in LPC4300 the mutex register is a new feature and the core-to-core interrupt structure has been improved.

The concrete implementation of a mailbox structure is very application specific and therefore goes beyond the goals of this application note. The basic mechanisms are explained in the following sections.



| Name | Access | Address offset | Description | Reset value |
|---|---|---|---|---|
| IRQ0 | R/W | 0x000 | Interrupt request register for the Cortex-M0+ CPU. | 0 |
| IRQ0SET | WO | 0x004 | Set bits in IRQ0 | - |
| IRQ0CLR | WO | 0x008 | Clear bits in IRQ0 | - |
| IRQ1 | R/W | 0x010 | Interrupt request register for the Cortex M4 CPU. | 0 |
| IRQ1SET | WO | 0x014 | Set bits in IRQ1 | - |
| IRQ1CLR | WO | 0x018 | Clear bits in IRQ1 | - |
| MUTEX | R/W [1] | 0x0F8 | Mutual exclusion register | 0x1 |

(1) Base address for the mailbox unit is 0x1C02 C000

**Fig 2. Registers associated with the mailbox unit**

## 3.2 Core-to-core interrupts

Each CPU can cause up to 32 user-defined interrupts to the other core. For this there are two 32-bit R/W registers IRQ0 and IRQ1 with their corresponding write-only SET and CLEAR registers.

**Table 1. IRQx, IRQxCLR and IRQxSET registers**

| | | | | | | |
|---|---|---|---|---|---|---|
| **IRQxCLR** | 31 | 30 | … | 1 | 0 | Write only |
| | Writing a bit in IRQxCLR **clears** corresponding bit in IRQx | | | | | |
| **IRQx** | 31 | 30 | … | 1 | 0 | Read/Write |
| | Writing a bit in IRQxSET **sets** corresponding bit in IRQx | | | | | |
| **IRQxSET** | 31 | 30 | … | 1 | 0 | Write only |

- Writing a ‘1’ into a bit of IRQ0 (either direct in IRQ0 or via IRQ0SET) fires an interrupt into direction Cortex-M0+.
- Writing a ‘1’ into a bit of IRQ1 (either direct in IRQ1 or via IRQ1SET) fires an interrupt into direction Cortex-M4F.

These 32 bits can be defined by the programmer to have a specific meaning.

**Example:**

- Data gets streamed in by the Cortex-M0+, further processing is done by the Cortex-M4F
- IRQ1 register is used to communicate the source address and number of bytes
- After the interrupt is processed by the Cortex-M4F it clears the IRQ1 register by writing 0xFFFF FFFF to IRQ1CLR

**Table 2. IRQ1 register used for address and buffer size information**

| Bit field usage | 16 bits for addressing a location in a buffer structure | 16 bits for the number of bytes available for processing |
|---|---|---|
| IRQ1 | 31 16 | 15 0 |

## 3.3 Mutex mechanism

The register MUTEX is intended to be used for a mutual exclusion mechanism. It can be accessed by both cores in atomic operation. When read for any reason, the current value will be returned and the bit will be cleared. The bit will be set again following any write.

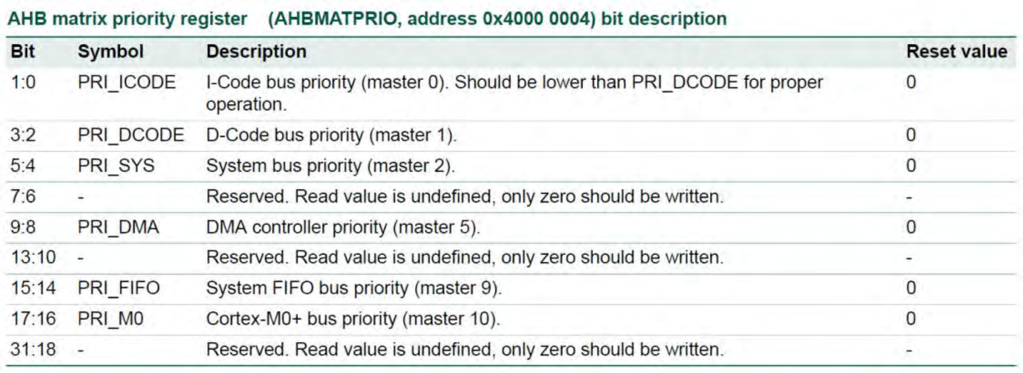
**Example:**

- The M4F wants to access a shared ring buffer structure.
- It reads the register and gets back a ‘1’, indicating that the mutex is free.
- By reading it, the bit gets cleared in order to show the resource allocation.
- If the Cortex-M0+ now reads the register it would see a ‘0’ and therefore needs to wait until it can read back a ‘1’.
- The ‘1’ needs to be set by the M4F when it wants to free up the mutex.

## 3.4 Shared RAM usage

As both cores have equal access to the chip resources, the internal SRAM can be used to create a communication pipeline in both directions. The access priority scheme on the AHB matrix can be influenced.

The Multilayer AHB Matrix arbitrates between several masters, only if they attempt to access the same matrix slave port at the same time. Care should be taken if the value in this register is changed, improper settings can seriously degrade performance. Priority values are 3 = highest, 0 = lowest. When the priority is the same, the master with the lower number is given priority. An example setting could put the Cortex-M4F D-code bus as the highest priority, followed by the I-Code bus. All other masters could share a lower priority.



**AHB matrix priority register (AHBMATPRIO, address 0x4000 0004) bit description**

| Bit | Symbol | Description | Reset value |
|---|---|---|---|
| 1:0 | PRI_ICODE | I-Code bus priority (master 0). Should be lower than PRI_DCODE for proper operation. | 0 |
| 3:2 | PRI_DCODE | D-Code bus priority (master 1). | 0 |
| 5:4 | PRI_SYS | System bus priority (master 2). | 0 |
| 7:6 | - | Reserved. Read value is undefined, only zero should be written. | - |
| 9:8 | PRI_DMA | DMA controller priority (master 5). | 0 |
| 13:10 | - | Reserved. Read value is undefined, only zero should be written. | - |
| 15:14 | PRI_FIFO | System FIFO bus priority (master 9). | 0 |
| 17:16 | PRI_M0 | Cortex-M0+ bus priority (master 10). | 0 |
| 31:18 | - | Reserved. Read value is undefined, only zero should be written. | - |

Fig 3. AHB matrix priority register

# 4. LPC54102 Dual Core Debugging

## 4.1 Dual core access with SWD

There is one Serial Wire Debug interface for both cores. The LPC54000 family does not support the JTAG interface for debugging, the interface can only be used for boundary scan.

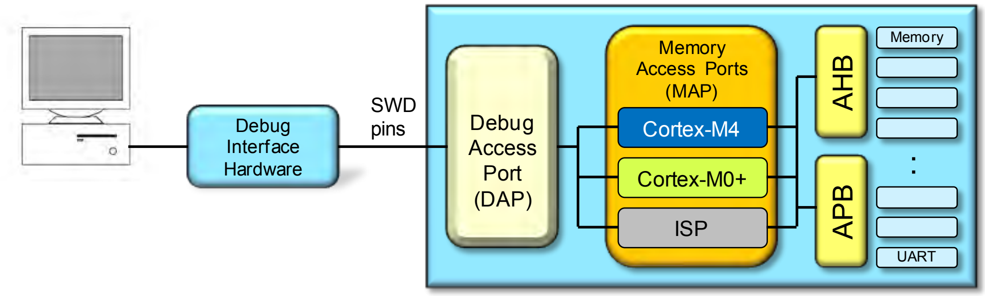


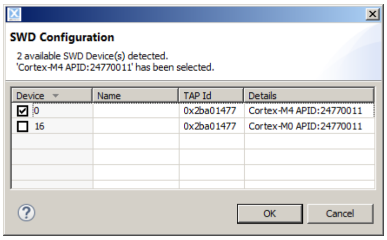
Fig 4. Debug Access Port (DAP) structure

One debug probe (for example LPC-Link2, Jlink, ULINK2, ULINKPro) can connect to both cores at the same time. The following two scenarios are possible:

- One SW debugger instance connects to one of the cores through one HW interface.
- Two SW debugger instances connect to both cores through one HW interface. For this the debugger software needs to know which Memory Access Port (MAP) needs to be addressed.
- One SW debugger to both cores

## 4.2 Debugging with LPCXpresso

- Version 7.5.0 of LPCXpresso fully supports the LPC54102 with its two cores.
- Please use the on-board Link2 debug interface configured for RedLink protocol (JP5 set and USB connection at J6).



**SWD Configuration**

2 available SWD Device(s) detected.
'Cortex-M4 APID:24770011' has been selected.

| Device | Name | TAP Id | Details |
|---|---|---|---|
| ☑ 0 | | 0x2ba01477 | Cortex-M4 APID:24770011 |
| ☐ 16 | | 0x2ba01477 | Cortex-M0 APID:24770011 |

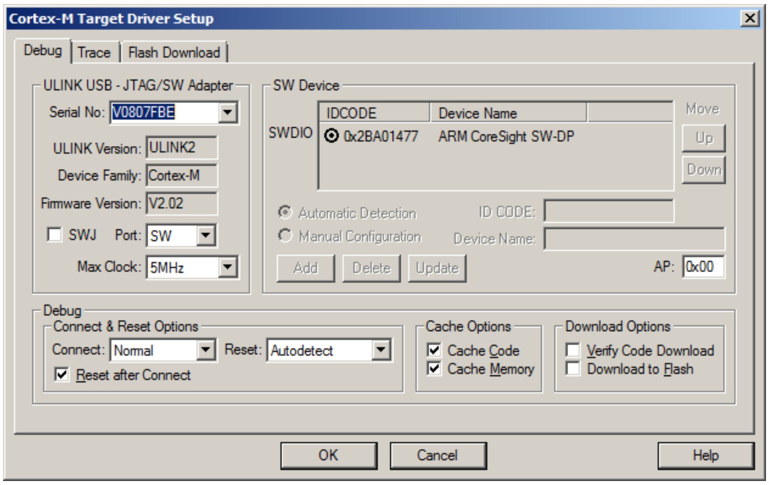
Fig 5. Memory AP selection with LPCXpresso

When the debugger starts up for the first time it detects the two cores and asks for selection. Once selected, the debugger keeps the setting for this debug configuration.

In the FAQ section for LPCXpresso on [www.lpcware.com](http://www.lpcware.com) a detailed description on how to enter a dual core debug session with LPC54102 is detailed.

### 4.3 Debugging with Keil µVision

- Keil µVision 5.xx fully supports the LPC54102 with its two cores.
- To use an external debugger, then any ULINK2/ME, ULINKPro, Jlink or CMSIS-DAP compatible debugger (also the Link2 board with CMSIS-DAP firmware) will work.



ULINK USB - JTAG/SW Adapter
Serial No: V0807FBE
ULINK Version: ULINK2
Device Family: Cortex-M
Firmware Version: V2.02
☐ SWJ Port: SW
Max Clock: 5MHz

SW Device

| | IDCODE | Device Name |
|---|---|---|
| SWDIO | ● 0x2BA01477 | ARM CoreSight SW-DP |

● Automatic Detection ID CODE:
○ Manual Configuration Device Name:
AP: 0x00

Debug
Connect & Reset Options
Connect: Normal Reset: Autodetect
☑ Reset after Connect

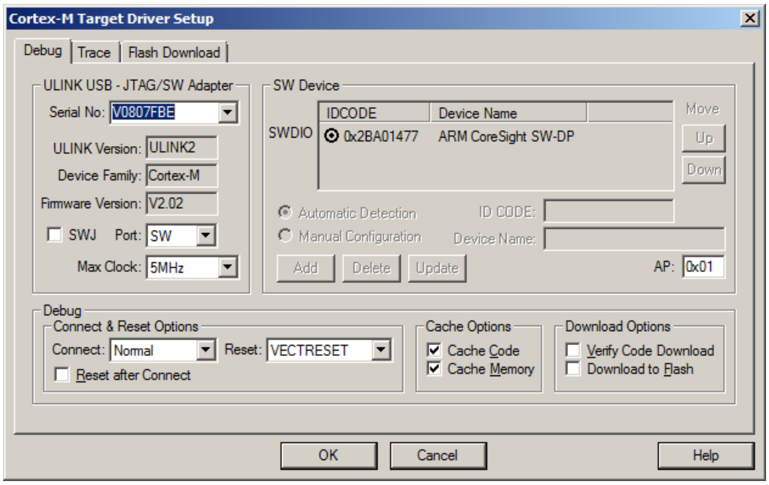
Cache Options
☑ Cache Code
☑ Cache Memory

Download Options
☐ Verify Code Download
☐ Download to Flash

(1) AP = 0x00 for a connection to the Cortex-M4F

(2) With “Reset after Connect” the chip is restarted after the debugger established a connection to the debug interface. This should provide a clean environment for the following debug session.

**Fig 6. Keil µVision: debugger settings and MAP selection for Cortex-M4F**



ULINK USB - JTAG/SW Adapter

Serial No: V0807FBE
ULINK Version: ULINK2
Device Family: Cortex-M
Firmware Version: V2.02
☐ SWJ Port: SW
Max Clock: 5MHz

SW Device

| | IDCODE | Device Name |
|---|---|---|
| SWDIO | ● 0x2BA01477 | ARM CoreSight SW-DP |

○ Automatic Detection
○ Manual Configuration
ID CODE:
Device Name:
AP: 0x01

Debug

Connect & Reset Options
Connect: Normal Reset: VECTRESET
☐ Reset after Connect

Cache Options
☑ Cache Code
☑ Cache Memory

Download Options
☐ Verify Code Download
☐ Download to Flash

(1) AP = 0x01 for a connection to the Cortex-M0+

(2) “Reset after Connect” must be unchecked, otherwise the chip starts over after the debugger connected to the Cortex-M0+ and the M0+ goes into reset state → debug connection lost

**Fig 7. Keil µVision: debugger settings and MAP selection for Cortex-M0+**

# 5. LPCOpen Dual Core Software Setup

## 5.1 Dual core project structure

Basically the projects for the Cortex-M4F and the Cortex-M0+ are separate projects. Each is compiled with its own settings for the respective instruction set. This means that at the end of the compilation for the two cores, there are two images which need to be programmed into flash. It is also possible to integrate one image into the other (as C-array source file or as AXF file) and program only one image.

Both methods come with pros and cons which will depend mostly on user preference.

### 5.1.1 Combined image download

There are a few different ways to integrate one image into the other. One is to generate a C-array from the linked code and put this as a source file into the other project. This is the most convenient method if one core executes from ROM and the other one from RAM. For Keil µVision projects, refer to the following example.

Example for Keil µVision:

- Cortex-M4F code is targeted to run from flash.
- Cortex-M0+ code is targeted to run from SRAM @ 0x0201 0000.
- The Cortex-M0+ project is compiled and linked first for internal SRAM, then the AXF file is converted into an array using the fromelf utility (part of KEIL µVision). The call can be added as a post-process (After Build/Rebuild) in the User tab

```
fromelf --cadcombined --output="<path>\M0_image_array.c" "<path>\CM0_RAM.axf”
```

- Now the Cortex-M4F project is compiled including the C file with the array as constant data.
- After linking the Cortex-M4F code to the flash memory region, the complete image can be programmed.
- At startup the Cortex-M4F copies the executable code for the Cortex-M0+ to the dedicated address 0x0201 0000.

The following is an example to use alternative utilities to generate this array.

**xd**: https://www.fourmilab.ch/xd/

The same principle would work with LPCXpresso. However, the example from LPCOpen uses directly the M0+ ELF (=AXF) file and provides it as input to the M4F project. The intermediate step with the conversion AXF → C-array is in principle redundant.

The relevant settings for a dual core project with LPCXpresso 7.5.0 can be found here:

*Properties → C/C++ Build → Settings → Multicore*

**Pro:**
- Only one image download to manage.
- In case of Cortex-M0+ execution from SRAM, the linker in the M4F project covers all code relocation jobs.

**Con:**
- A change in the Cortex-M0+ code requires a recompilation of the M4F software.
- Relocation of the Cortex-M0+ code takes time (longer startup phase).

In case both images are executed from the same flash memory, then the two separate images can be combined with a utility and then be programmed as one image.

### 5.1.2 Separate image downloads

If both code images reside in flash memory, they are normally compiled and programmed separately. The only hard dependency between the two projects is the start address of the Cortex-M0+ image, this is something the Cortex-M4F code needs to know.

Example for Keil µVision:

- Cortex-M4F code is targeted to run from flash @ 0x0000 0000.
- Cortex-M0+ code is targeted to run from flash @ 0x0002 0000.
- The Cortex-M4F and the Cortex-M0+ projects are compiled and linked and then programmed separately to their dedicated flash memory addresses.
- At startup the Cortex-M4F initializes the stack pointer and start address of the Cortex-M0+ code and then takes the Cortex-M0+ out of reset.

The same principle would work with LPCXpresso.

| | | |
|---|---|---|
| **Pro:** | - | Independent projects, compilations and downloads |
| **Con:** | - | Two image downloads to manage |

## 5.2 Boot process

After a power-up or hardware reset, the boot process is handled by the on-chip boot ROM and is always executed by the Cortex-M4F core. After the bootloader code and the system init section, the Cortex-M4F can set up the environment for the Cortex-M0+.

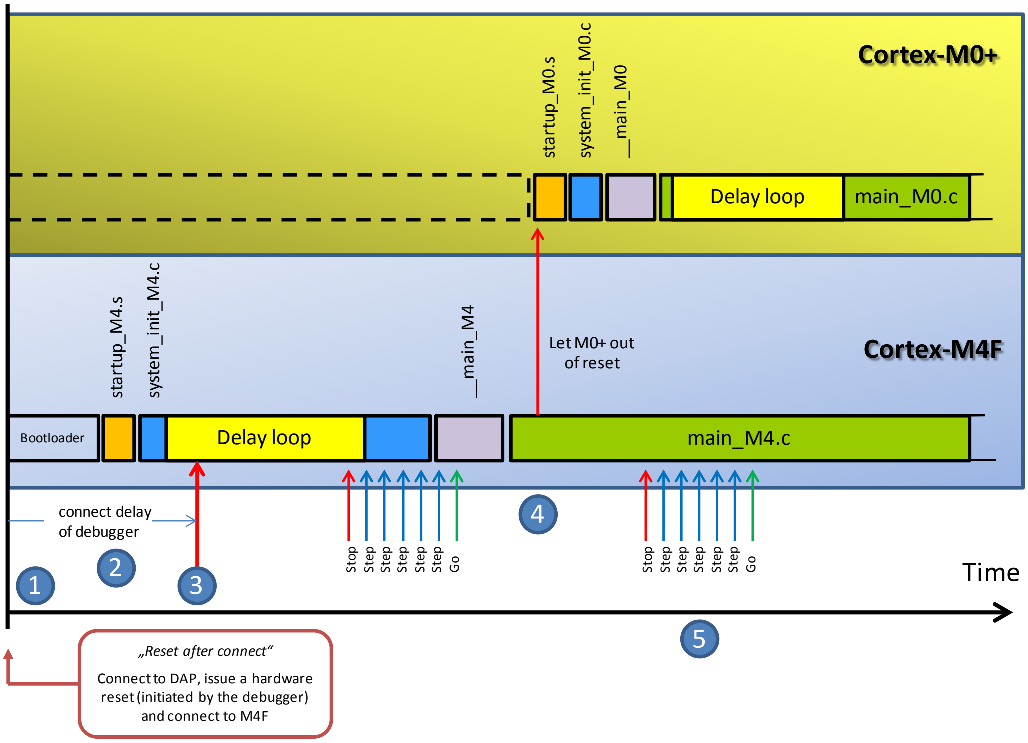
- Provide a reset handler start address to the Cortex-M0+
- Provide a stack pointer address to the Cortex-M0+

After that the Cortex-M0+ can be taken out of reset by the Cortex-M4F.

Note: It is very important to understand how the boot process works, in order to maneuver around the various issues which can arise in a dual core project and during programming and debugging.

The setups shown in the following sections 5.2.1, 5.2.2 and 5.2.3 are examples of how such projects can be implemented and handled. For example it is not mandatory to work with soft delay loops at the beginning of the code. This is one possibility how to work around some foibles of the dual core implementation and the behavior of debugger connections in general.

### 5.2.1 Boot process when connecting with a debugger to the Cortex-M4F



(1) This flowchart shows a suggested safe way to help with debugger connections. The software examples for this application note do not implement the delay loops.

**Fig 8. Boot process when connecting a debugger to the Cortex-M4F**

(1) After the debugger is connected to the Debug Access Port it applies a reset signal to the MCU ( *“Reset after Connect”* ).

(2) During the connect delay of the debugger the on-chip bootloader and the system init software starts to run.

(3) After some unspecified period (debugger dependent) the debugger stops the M4F core. In the figure this happens in a delay loop. This loop is a simple soft loop (for example 2 seconds) that prevents the code to run into a more sophisticated application code like an OS, before the debugger is able to get to the core and stop it. From this point stepping through the Cortex-M4F code with the debugger is possible.

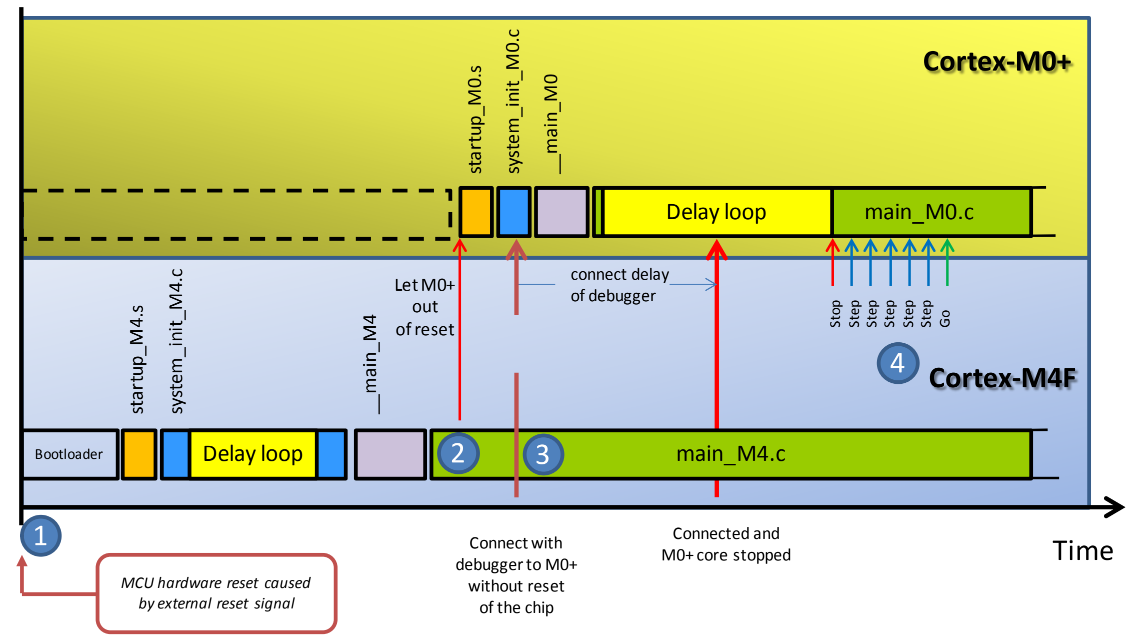
(4) The Cortex-M4F takes the Cortex-M0+ out of reset. From this point in time the Cortex-M0+ executes its application code. The delay loop in the Cortex-M0+ code is also just there to give the user the possibility to access in a clean way during this period. (see chapter 5.2.2)

(5) The Cortex-M4F is still under full control of the debugger, but there is no control over the M0+.

**Notes:**

- Debuggers can also set breakpoints in advance (“Run to main”), then the delay loop for the M4F is not required because it will stop before the application code anyway. But with the delay loop, which could be even placed before SystemInit, the boot process are really under “manual” control.
- The software shown in the figure is similar to CMSIS. If the chip already prevents SWD access with a setting done in system_init (or even crashes) then the delay loop in main() does not have any effect. In a development phase where the system_init is subject to heavy changes, it is good to have this delay loop already at the beginning of system_init.
- The delay loop becomes more important when one wants to flash the device during software development and the application code somehow prevents this because of certain settings or even hangs. This problem can be solved by going into ISP mode, but maybe this pin is not available on the hardware. Then the delay loop is the solution. It should be long enough to let the debugger connect to the core after the flash download is initiated.

### 5.2.2 Boot process when connecting with a debugger to the Cortex-M0+



(1) This flow shows a suggested way to help debugger connections. The software examples for this application note do not implement the delay loops.

Fig 9. Boot process when connecting a debugger to the Cortex-M0+

(1) The user initiates a HW reset for the MCU. The debugger can do this job as well, but the timing of this signal is hard to predict. It cannot be ensured that the debugger access to the Cortex-M0+ core happens at the right time between 2 and 3.

(2) After the Cortex-M4F code prepared the Cortex-M0+ and took it out of reset, the Cortex-M0+ is accessible and starts running independent of from the Cortex-M4F core. In order to recognize this point in time it is good practice to switch on an

LED to mark this transition step. Alternatively an estimation on the length of the Cortex-M4F delay loop is needed in order to initiate the debugger connection at the right time

(3) The debugger can now access the Cortex-M0+. This needs to be done without a reset signal to the rest of the chip. Otherwise the MCU starts over, the Cortex-M0+ goes back to reset state and is no longer accessible until the Cortex-M4F takes it out of reset again. Most debuggers do not accept this connection breakdown and issue an error. As this debugger access takes some unspecified time, the delay loop should be long enough to prevent the Cortex-M0+ from running already in more sophisticated application code.

(4) After the debugger connected to the core and stopped it in the delay loop the user can step through the Cortex-M0+ application code. This does not affect the activities on the Cortex-M4F side.

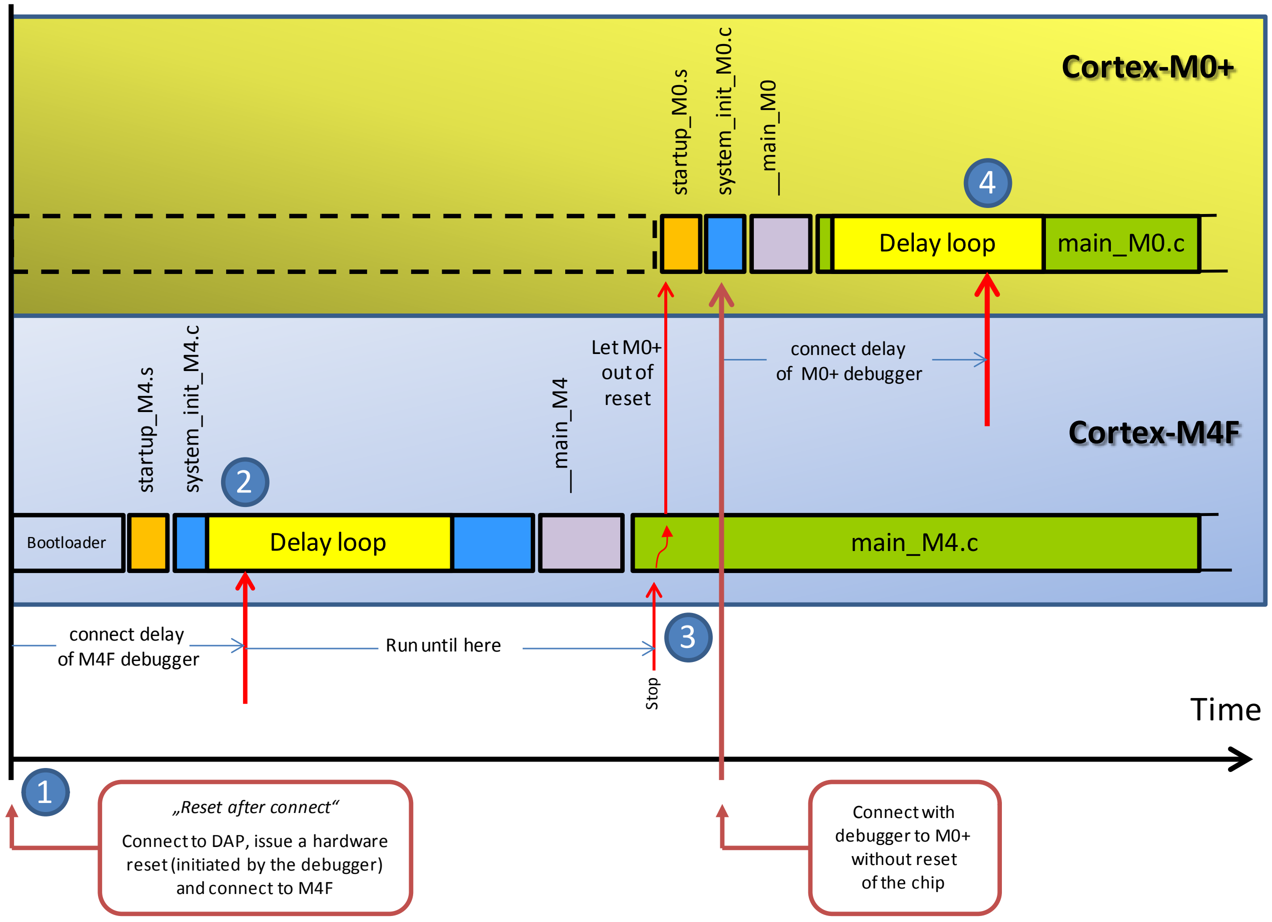
**Notes:**

- For just the Cortex-M0+ debugger access the delay loop on the Cortex-M4F side could be removed, but for the previously mentioned reasons (e.g. Cortex-M4F SW crashes) it is really useful to implement these delays during the software development phase. This could be conditional by asking for a level on a GPIO, 0 = no loops and 1 = loops.
- Reset options in the debugger settings are defined differently. For the Cortex-M0+ side it is important that there is no reset performed. Therefore, choose a configuration which incorporates “no reset”.

### 5.2.3 Boot process when connecting with two debuggers to the LPC5410x

This section briefly describes what should be considered if concurrently debugging the Cortex-M4F and the Cortex-M0+.

- The connection via SWD is quite unstable with the Link2 (CMSIS-DAP firmware) and the ULLINK2/ME.
- ULINKPro provides a stable connection to the two cores.
- LPCXpresso + Link2 (on-board and external) with RedLink firmware
- With Keil µVision or IAR EWARM then two debugger instances are needed by starting the program twice. Then both debug views are visible at the same time.
- For LPCXpresso v7.5.0, run with one instance and with some additional configuration the debug view for both cores are visible.
- Connect to the cores in a specific order.



(1) This flow shows a suggested way to help debugger connections. The software examples for this application note do not implement the delay loops.

Fig 10. Boot process when connecting a debugger to the Cortex-M4F and Cortex-M0+

(1) After the M4F debugger instance connects to the Debug Access Port, it applies a reset signal to the MCU. During the connect delay of the debugger the on-chip bootloader and the system init software start to run.

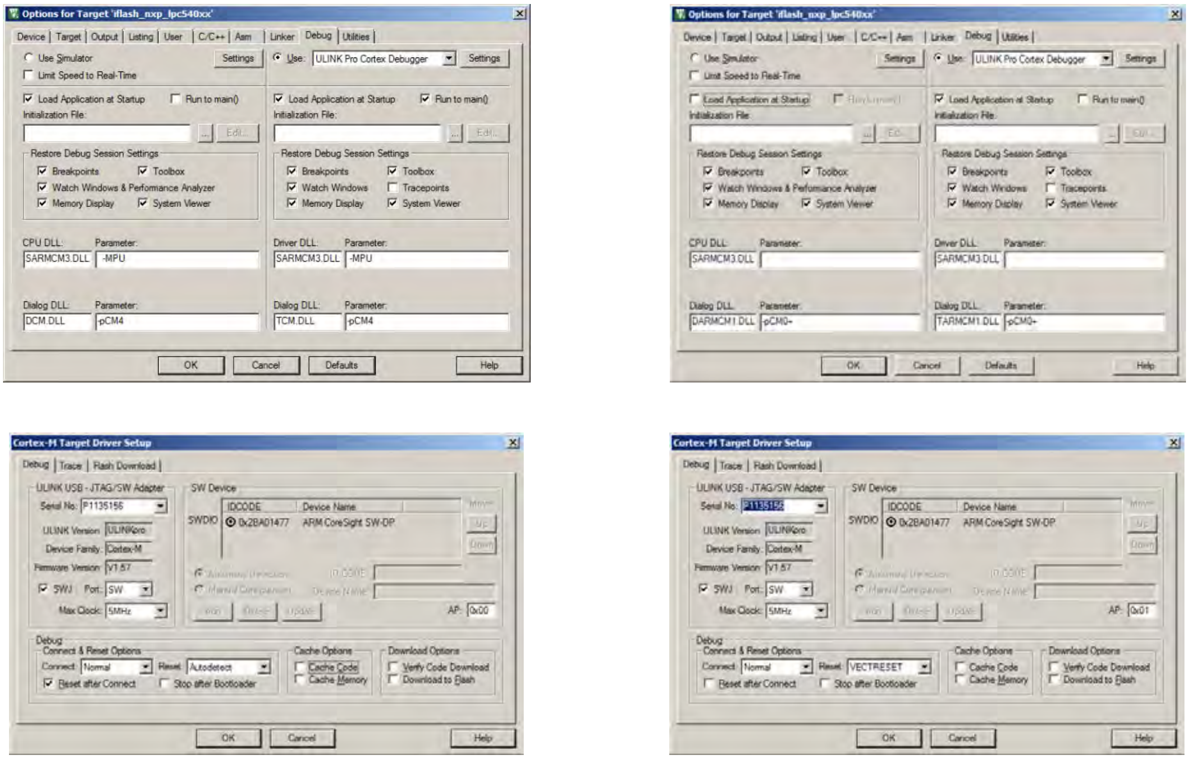
(2) After some unspecified period (debugger dependent) the debugger grabs the Cortex-M4F and stops it. In the figure this happens in the delay loop. This loop is a simple soft loop (for example 2 seconds) which prevents the code to run into more sophisticated application code like the OS. This could cause problems with the debugger connection (very much debugger dependent).

(3) Execute to the point where the Cortex-M0+ is taken out of reset. Prepare the Cortex-M0+ debugger instance in such a way that it is ready to connect. Then take the Cortex-M0+out of reset with the Cortex-M4F debugger and connect immediately the Cortex-M0+ debugger instance to the core.

(4) After the connect delay the debugger grabs and stops the Cortex-M0+ (still in the delay loop).

#### 5.2.3.1 Configuration example for Keil µVision + ULINKPro

For a dual core debug session with µVision 5.12 and ULINKPro please use the following settings:



Options for Target 'iflash_nxp_lpc540xx' – Debug (Cortex-M4F):
○ Use Simulator ☐ Limit Speed to Real-Time
● Use: ULINK Pro Cortex Debugger
☑ Load Application at Startup ☑ Run to main()
Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watch Windows ☐ Tracepoints ☑ Memory Display ☑ System Viewer
CPU DLL: SARMCM3.DLL Parameter: -MPU
Dialog DLL: DCM.DLL Parameter: -pCM4
Driver DLL: SARMCM3.DLL Parameter: -MPU
Dialog DLL: TCM.DLL Parameter: -pCM4

Cortex-M Target Driver Setup – Debug (Cortex-M4F):
Serial No: P1135156
ULINK Version: ULINKpro
Device Family: Cortex-M
Firmware Version: V1.57
☑ SWJ Port: SW
Max Clock: 5MHz
SWDIO ● 0x2BA01477 ARM CoreSight SW-DP
AP: 0x00
Connect: Normal Reset: Autodetect
☑ Reset after Connect ☐ Stop after Bootloader
☐ Cache Code ☐ Cache Memory
☐ Verify Code Download ☐ Download to Flash

Options for Target 'iflash_nxp_lpc540xx' – Debug (Cortex-M0+):
● Use: ULINK Pro Cortex Debugger
☑ Load Application at Startup ☐ Run to main()
Restore Debug Session Settings: ☑ Breakpoints ☑ Toolbox ☑ Watch Windows ☐ Tracepoints ☑ Memory Display ☑ System Viewer
CPU DLL: SARMCM3.DLL Parameter:
Dialog DLL: DARMCM1.DLL Parameter: -pCM0+
Driver DLL: SARMCM3.DLL Parameter:
Dialog DLL: TARMCM1.DLL Parameter: -pCM0+

Cortex-M Target Driver Setup – Debug (Cortex-M0+):
Serial No: P1135156
ULINK Version: ULINKpro
Device Family: Cortex-M
Firmware Version: V1.57
☑ SWJ Port: SW
Max Clock: 5MHz
SWDIO ● 0x2BA01477 ARM CoreSight SW-DP
AP: 0x01
Connect: Normal Reset: VECTRESET
☐ Reset after Connect ☐ Stop after Bootloader
☐ Cache Code ☐ Cache Memory
☐ Verify Code Download ☐ Download to Flash

a. Settings for Cortex-M4F

b. Settings for Cortex-M0+

**Fig 11. Dual Core debug settings for ULINKPro**

(1) Connect the LPC54102 board to 5V USB power using J4.

(2) Connect a ULINKPro to the SWD connector P1.

(3) Start µVision two times and load the example described in chapter 6.2 two times. In one µVision instance select the m4_blinky as active and in the other m0_blinky.

(4) Apply the debug settings shown in Fig 11.

(5) Start the Cortex-M4F debugger and start the application. This means that the M4F also prepares the M0+ and starts it. The LED blinking is visible and if connected to a terminal debug output also.

(6) Start the Cortex-M0+ debugger. The application stops because the Cortex-M0+ gets halted by the debugger at its current program counter address.

(7) Now debug is possible, start/stop on the two cores, define breakpoints etc.

#### 5.2.3.2 Configuration example for LPCXpresso + Link2

LPCXpresso can connect to the two cores using the RedLink debug server. On the following website a step-by-step description is provided:

http://www.lpcware.com/content/faq/lpcxpresso/lpc43xx-multicore-apps

## 5.3 Troubleshooting

It is difficult to provide a robust debugging methodology because of the various combinations of software and hardware tools along with many possible states of the dual core platform. Iterating and analyzing results till a working solution is achieved is the best approach.

Most debug problems have quite trivial reasons on the MCU side:

- Enabled watchdog.
- A disabled core.
- Transition into a Power-down mode.

On the debugger side the issues could be:

- A wrong reset strategy.
- Clock frequency too high.
- Wrong MAP selection.
- Limited capabilities of the debugger firmware.
- With the limited bandwidth of the SW interface one should be cautious with the debugger windows. An open window with a memory view, for example, causes the debugger to catch this information from the MCU. This can not only lead to slow processing and screen updates when stepping through the code, but also a change in the real time behavior due to bus arbitrations.

# 6. Dual Core Software example

The example is an enhanced version of the multicore example (m4master_blinky / m0slave_blinky) one can find in the LPCOpen software package for the LPC54102.

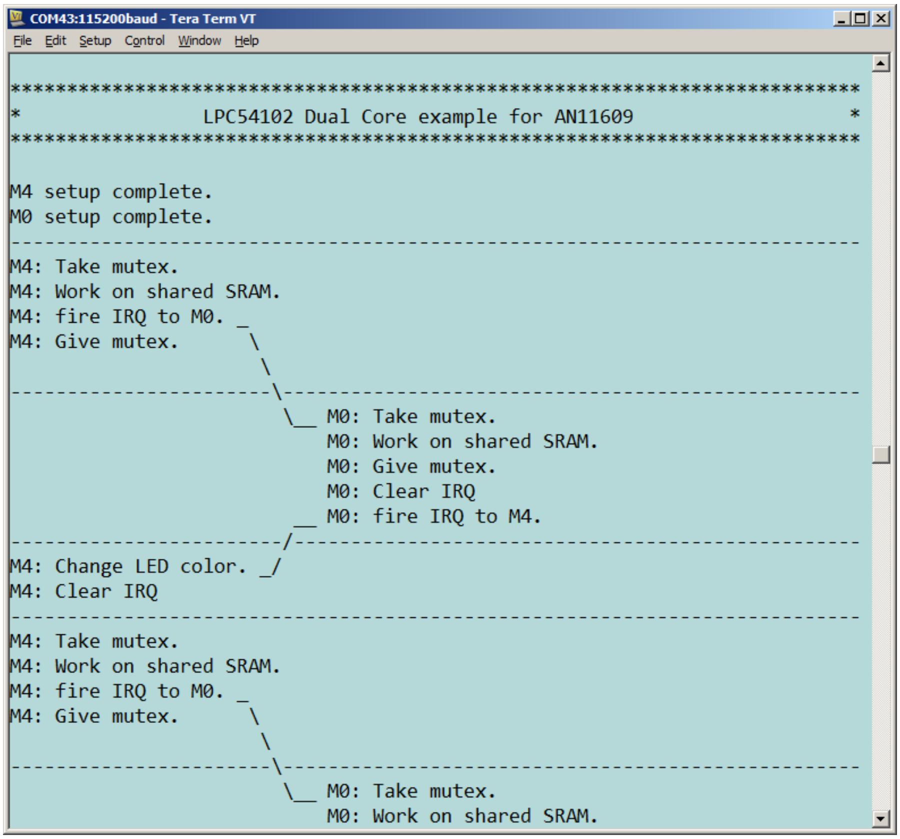
http://www.lpcware.com/content/nxpfile/lpcopen-platform

The example is provided for LPCXpresso version 7.5.0 and for Keil µVision 5.12 (with Device Family Pack for LPC54xxx).

It shows the three mechanisms described earlier:

1. Core-to-core interrupt.
2. Use of the hardware mutex register.
3. Work with both cores on a shared SRAM location.

In addition to the blinking LED (red-green-red-green), the program uses the UART as output for visualizing the flow. To connect the UART to a PC connect a RS232 transceiver to J5, e.g. a UART-2-USB cable.


COM43:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
***************************************************************************
*                LPC54102 Dual Core example for AN11609                   *
***************************************************************************

M4 setup complete.
M0 setup complete.
---------------------------------------------------------------------------
M4: Take mutex.
M4: Work on shared SRAM.
M4: fire IRQ to M0. _
M4: Give mutex.      \
                      \
-----------------------\---------------------------------------------------
                        \__ M0: Take mutex.
                            M0: Work on shared SRAM.
                            M0: Give mutex.
                            M0: Clear IRQ
                         __ M0: fire IRQ to M4.
------------------------/--------------------------------------------------
M4: Change LED color. _/
M4: Clear IRQ
---------------------------------------------------------------------------
M4: Take mutex.
M4: Work on shared SRAM.
M4: fire IRQ to M0. _
M4: Give mutex.      \
                      \
-----------------------\---------------------------------------------------
                        \__ M0: Take mutex.
                            M0: Work on shared SRAM.
```

Fig 12. Debug output for LPC54102 dual core example

## 6.1 LPCXpresso project

The LPCXpresso v7.5.0 project is configured this way:

- Cortex-M4F software runs from flash starting at 0x0000 0000
- The Cortex-M0+ code is linked for SRAM at 0x0201 0000 and integrated into the Cortex-M4F image → the flash download of the Cortex-M4F project contains the Cortex-M0+ code

Follow these steps to compile and load the software example:

1. Connect USB port J6 of an LPCXpresso LPC54102 board to the PC (JP5 set, JP7 set to 2-3, JP2 set to 1-2)
2. Import all projects of the ZIP file *AN11609_Example_LPCXpresso.zip* into the project manager using the "*Import project(s")* button from the quickstart panel.
3. Compile all projects with *Project → Build All*
4. Select the *multicore_m4master_blinky* and program it using the ***Program flash*** icon. Select the file to download.
   *\multicore_m4master_blinky\Debug\multicore_m4master_blinky.axf*
5. Starting the debugger with the Debug button from the quickstart panel is also an option. This compiles and flashes the code and starts the debugger.
6. On J5, can connect a UART-2-USB cable to see the debug output on a terminal (115200 baud, 8N1).

## 6.2 Keil µVision project

The µVision v5.12 project is configured this way:

- Cortex-M4F software runs from flash starting at 0x0000 0000
- The Cortex-M0+ code runs from flash starting at 0x0002 0000

Follow these steps to compile and load the software example:

1. Connect USB port J6 of a LPCXpresso LPC54102 board to the PC (JP5 not set, JP7 set to 2-3, JP2 set to 1-2).
   Make sure to firstly download and install the CMSIS-DAP firmware.
2. Open the multi-project file
   *.\applications\lpc5410x\keil\lpcxpresso_54102\lite_examples.uvmpw*
3. Compile all projects with *Project → Batch Build → Select All → Rebuild*
4. Set *m4_blinky* as active (right mouse click on project) and download it to flash with the icon *Download* (or *Flash → Download*).
5. Set *m0_blinky* as active and download it to flash.
   Note: for this example the MAP has been set to 0x00 in order to use the M4 for the download. As explained in chapter 5.2, the Cortex-M0+ is not always accessible by the debugger and therefore the flash download could fail. If one wants to debug the Cortex-M0+ code, then a change of settings is required according to Fig 7.
6. On J5, connect a UART-2-USB cable to see the debug output.

# 7. Conclusion

The software example provided in this Application note illustrates the basic mechanisms for the implementation of a mailbox structure between the two cores of the LPC54102.

A more generic implementation for the LPC4300 can be found in this application note:

http://www.lpcware.com/content/nxpfile/an11177-inter-processor-communication-lpc43xx

In a real application the focus is normally on “efficient” or “fast”, not on “generic”. The data flow is sometimes unidirectional, latency must be minimized, only little SRAM available, these are all aspects which heavily influence the design of a communication interface between the cores.

# 8. Legal information

## 8.1 Definitions

**Draft —** The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

## 8.2 Disclaimers

**Limited warranty and liability —** Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes —** NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use —** NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications —** Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Export control —** This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Evaluation products —** This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

## 8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

# 9. List of figures

# 10. Contents

For more information, visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

**Date of release: 4 November 2014**
**Document identifier:AN11609**