Changing Exception level and Security state in an embedded image

Version 1.0

Non-Confidential

Copyright © 2021 Arm Limited (or its affiliates).
All rights reserved.

Issue 04

102437_0100_04_en

Changing Exception level and Security state in an embedded image

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0100-04	1 January 2021	Non-Confidential	Initial release

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at https://www.arm.com/company/policies/trademarks.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Contents

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1. Overview

This guide is the fourth in a collection of related guides:

- Building your first embedded image
- Retargeting output to UART
- Creating an event-driven embedded image
- Changing Exception level and Security state in an embedded image (this guide)

In the previous guides, we built an Executable and Linkable Format (ELF) image to expose some features of the Armv8-A architecture and toolchain for embedded software development. We printed hello world to a Telnet console, and enabled interrupts on the system.

In this guide, we discuss the architectural features of Exception level and Security state in more detail. At the end of this guide, you will understand how to use exceptions to move through different exception levels and switch between the Secure and Non-secure worlds.

Before you begin

To complete this guide, you will need to have Arm Development Studio Gold Edition installed. If you do not have Arm Development Studio, you can download a 30-day free trial.

Arm Development Studio Gold Edition is a professional quality tool chain developed by Arm to accelerate your first steps in Arm software development. It includes both the Arm Compiler 6 toolchain and the FVP_Base_Cortex-A73x2-A53x4 model that are used in this guide. We will use the command-line tools for most of the guide. This means that you will need to configure your environment in order to run Arm Compiler 6 from the command-line.

The individual sections of this guide contain some code examples. These code examples are available to download as a ZIP file:

CommonTasks-ChangingExceptionLevelsAndSecurityState.zip

If you want to use the Arm Development Studio GUI instead of the command line tools, follow the instructions in Arm Development Studio Getting Started Guide, Tutorial: Hello World.

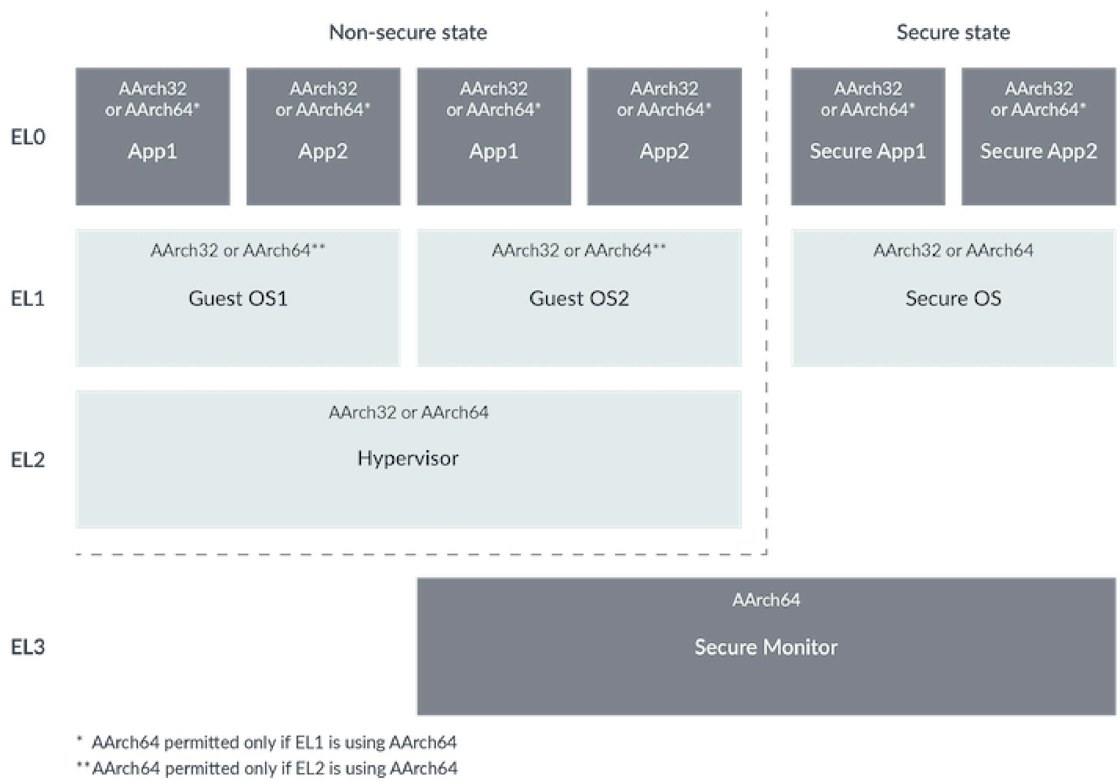
Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

2. Exception levels

The Armv8-A architecture includes a series of Exception levels, which have different privileges to control system register accessibility and instruction availability.

These different Exception levels determine what a hardware component can do at a specific time. For example, an operating system running at a higher Exception level has access to more features than user software running at a lower Exception level. This means that user software can be prevented from carrying out certain actions and accessing certain features.

Figure 2-1: Changing exception level and security diagram



EL3 is the most privileged Exception level. The other Exception levels are built upon EL3 in the abstraction stack. For example, the Architectural Feature Trap Register (EL3), CPTR_EL3, is only accessible at EL3. This register controls a few things, including floating-point operations, for all Exception levels. A similar register at EL2, CPTR_EL2, is accessible at EL2 and EL3. However, CPTR_EL2 only affects EL2 and lower Exception levels. The main purpose of this hierarchy is to grant each piece of software only as much control as it needs. This is because you would not want a user application to have the same level of system control as an operating system.

Exception level is just one factor in determining which privileges are granted. Security state and Execution state are also factors. Security state controls access to certain registers and memory regions that are marked as Secure. The Execution state of the processor can be either 32-bit or 64-

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

bit. We will not discuss those differences in this guide. The relationship between Security state and Exception level is complex. For the purpose of this guide, you should know that:

- There is no Secure EL2 in the architecture.
- There is no distinction between Security states at EL3.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Changing Exception levels

Switching between Exception levels is done by returning from an exception. However, we must also have something to return to. In `startup.s` we define a function:

```
.global el1_entry_aarch64
    .type el1_entry_aarch64, "function"

el1_entry_aarch64:
  // we can use the same vector table in this example, but in general
  // each combination of Exception level, Security state, and Execution state
  // will need a new vector table
  LDR      x0, =vectors
  MSR      VBAR_EL1, x0

  //we must ensure that floating point register accesses are not trapped
  //since the c library for AArch64-v8A uses them
  MOV      x0, #(0x3 << 20)
  MSR      CPACR_EL1, x0

  // ensures that all instructions complete before
  ISB
  // Branch to scatter loading and C library init code
  .global  __main
  B        __main
```

We now have a label that the `ERET` can branch to. This means that we can modify the `start64` function, as shown in the following code:

```
boot:
  ADRP x0, Image$$STACK_EL3$$ZI$$Limit // get stack address
  MOV  sp, x0

  // NB, CODE OMITTED

  // Configure SCR_EL3
  // ------------------
  MOV      w1, #0              // Initial value of register is unknown
  ORR      w1, w1, #(1 << 11)  // set ST bit (disable trapping of timer control
 registers)
  ORR      w1, w1, #(1 << 10)  // set RW bit (next lower EL in aarch64)
  ORR      w1, w1, #(1 << 3)   // Set EA bit (SError routed to EL3)
  ORR      w1, w1, #(1 << 2)   // Set FIQ bit (FIQs routed to EL3)
  ORR      w1, w1, #(1 << 1)   // Set IRQ bit (IRQs routed to EL3)
  MSR      SCR_EL3, x1

  // NB, CODE OMITTED

  // Initialize SCTLR_EL1
  // --------------------
  // SCTLR_EL1 has an unknown reset value and must be configured
  // before we can enter EL1
  MSR      SCTLR_EL1, xzr

  LDR      x0, =el1_entry_aarch64
  LDR      x1, =AArch64_EL1_SP1
  MSR      ELR_EL3, x0 // where to branch to when exception completes
  MSR      SPSR_EL3, x1 // set the program state for this point to a known value

  BL       gicInit

  ERET
```

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

In the preceding code, the following operations are performed:

1. Define a new stack pointer for the current exception level. In the previous guides (Building your first embedded image, Retargeting embedded output to UART, and Creating an event-driven embedded image), we relied on the Arm C libraries to initialize the stack pointer. Because we have moved our branch to `__main`, this will only initialize a stack pointer for EL1. We also add the line`STACK_EL3 +0 ALIGN 64 EMPTY 0x4000 {}` in `scatter.txt` to define the stack in memory.
2. Disable trapping of the timer register accesses, because the processor will be in EL1 when the timer interrupt is generated.
3. Set the next lower Exception level, Secure EL1, to the 64 bit Execution state.
4. Ensure the System Control Register, SCTLR_EL1, is zero initialized, and set the Exception Link Register, ELR_EL3, and Saved Program State Register, SPSR_EL3, to the desired address and state at EL1.

Note

SPSR_EL3 is responsible for controlling the Exception level that the processor enters after the ERET, while ELR_EL3 merely specifies the address to return to.

---

5. Move the branch to `gicInit` here. Because this function modifies registers accessible at EL3 only, it cannot be placed in the `main()` function, because that function is now at EL1.

Building and running this code will send a hello world and interrupt message, as we saw in the previous guides in this series.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4. Security state

The Arm Architecture Reference Manual introduces a Secure state and a Non-secure state for the processor.

- Secure state can access Secure and Non-secure physical addresses.
- Non-secure state can only access the Non-secure address space and cannot access certain Secure system registers.

Partitioning memory accesses into Secure state and Non-secure state prevents, for example, a user level application in Non-secure EL0 from accessing encryption keys held by a trusted operating system running in Secure EL1. This partitioning of memory accesses is also important for the implementation of Arm TrustZone technology.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5. Switching Security state

Control over Security state is performed at EL3, which sets the Security state of lower exception levels.

Specifically, setting the leading bit of the Secure Configuration Register SCR_EL3, will put the system into a Non-secure state, after the system returns to a lower exception level. However, this is not the only change that you will have to make, because the Non-secure state introduces some complexities. Follow these steps to ensure that any instructions that are executed while they are in Non-secure state are in Non-secure memory:

1. Modify `scatter.txt`:

```
SROM_LOAD 0x00000000
  {
    SROM_EXEC +0
    {
      startup.o(BOOT, +FIRST)
          gic.o
    }

    STACK_EL3 0x04000000 ALIGN 64 EMPTY 0x10000 {}
  }

  NSROM_LOAD 0x80000000
  {
    ROM_EXEC +0 0x10000
    {
      startup.o(NONSECURE)
      * (+RO)
       }

    RAM_EXEC +0 0x10000
    {
      * (+RW, +ZI)
    }

    ARM_LIB_STACKHEAP +0 ALIGN 64 EMPTY 0x10000 {}

    STACK_EL2 +0 ALIGN 64 EMPTY 0x10000 {}
  }
```

   This scatter file defines a new region of memory, `NSROM_LOAD`, starting at the Non-secure DRAM portion of the memory in the model. The `NONSECURE` section of our startup code, which you will define later, is placed in this region. Wildcard data has been placed in this region, so that all data which is not placed elsewhere will be placed in the relevant regions here. You have also defined a stack for EL2, and moved the library stack-heap here. The `SROM_LOAD` region is located in Secure memory, and the `gic.o` and `BOOT` sections of the code are also in this region. The EL3 stack has been placed in Secure SRAM.

2. Because the code branches to `__main` in Non-secure EL1, you must change references to the Secure timer registers to the Non-secure timer registers. Modify `timer.s` to replace accesses to CNTPS_TVAL_EL1 and CNTPS_CTL_EL1 with CNTP_TVAL_EL0 and CNTP_CTL_EL0.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Define the EL1 and EL2 entry functions in `startup.s`, and wrap them into the section named `NONSECURE`, so that they are placed in Non-secure memory.

```
// ------------------------------------------------------------
// EL2 AArch64
// ------------------------------------------------------------

    .section NONSECURE, "ax"
    .align 3

    .global el2_entry_aarch64
    .type el2_entry_aarch64, "function"
el2_entry_aarch64:
  NOP
  ADRP x0, Image$$STACK_EL2$$ZI$$Limit
  MOV  sp, x0
  // Configure HCR_EL2 - the hypervisor configuration register
  // ------------------------------------------------------------
  NOP
  MRS      x0, HCR_EL2
  MOV      x1, #(1 << 31)
  ORR      x0, x0, x1
  MSR      HCR_EL2, x0

  // Configure CNTHCTL_EL2 - the Counter-timer Hypervisor Control register
  // ------------------------------------------------------------------------
  // Enable timer register access for lower EL levels
  MRS      x0, CNTHCTL_EL2
  ORR      x0, x0, #(1 << 1)
  ORR      x0, x0, #1
  MSR      CNTHCTL_EL2, x0

  // Possible to use the same vector table in this example, but in general
  // each combination of Exception level, Security state, and Execution state
  // will need a new vector table
  // ADD YOUR CODE HERE
  LDR      x0, =vectors
  MSR      VBAR_EL2, x0

  // Initialize SCTLR_EL1
  // --------------------
  // SCTLR_EL1 has an unknown reset value and must be configured
  // before entering EL1
  MSR      SCTLR_EL1, xzr

  // Enter EL1
  // ---------
  LDR      x0, =el1_entry_aarch64
  LDR      x1, =AArch64_EL1_SP1
  MSR      ELR_EL2, x0
  MSR      SPSR_EL2, x1
  ERET


// ------------------------------------------------------------
// EL1 AArch64
// ------------------------------------------------------------
    .global el1_entry_aarch64
    .type el1_entry_aarch64, "function"
el1_entry_aarch64:
  // Can use the same vector table in this example, but in general
  // each combination of Exception level, Security state, and Execution state
  // will need a new vector table
  LDR      x0, =vectors
  MSR      VBAR_EL1, x0

  //Ensure that floating point register accesses are not trapped
  //since the c library for AArch64-v8A uses them
  MOV      x0, #(0x3 << 20)
```

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

```
  MSR      CPACR_EL1, x0

  // ISB ensures that all instructions complete before this instruction
  ISB
  // Branch to scatter loading and C library init code
  .global   __main
  B         __main
```

4. The comments in the code explain the modifications to system registers. It is not necessary to change exception level incrementally. Configuration of the registers in `el2_entry_aarch64` could have been done at EL3, to return from EL3 directly to EL1. Instead, the state of EL1 is configured at EL2. Now that the entry points have been defined, let's turn our attention to the interrupt controller. In `gic.s`:

```
MOV      x0, #ICC_SRE_ELn.Enable
ORR      x0, x0, #ICC_SRE_ELn.SRE
MSR      ICC_SRE_EL3, x0
ISB
MSR      ICC_SRE_EL2, x0
ISB
MSR      ICC_SRE_EL1, x0
// Set the Secure version of ICC_SRE_EL1
ISB
MRS      x1, SCR_EL3
BIC        w1, w1, #1          // Set NS bit (lower EL in Secure state)
MSR      SCR_EL3, x1
ISB
MSR      ICC_SRE_EL1, x0
MRS      x1, SCR_EL3
ORR        w1, w1, #1          // Set NS bit (lower EL in non Secure state)
MSR      SCR_EL3, x1
ISB
MOV      x0, #0xFF
MSR      ICC_PMR_EL1, x0 // Set PMR to lowest priority
ISB
MOV      x0, #3
MSR      ICC_IGRPEN1_EL3, x0
ISB
MOV      x0, #1
MSR      ICC_IGRPEN1_EL1, x0
MSR      ICC_IGRPEN0_EL1, x0
ISB
```

5. Build the image, then run the model:

```
$ FVP_Base_Cortex-A73x2-A53x4 -C bp.refcounter.non_arch_start_at_default=1 -a
 __image.axf
```

This generates the same Telnet messages that we saw in the Retargeting output to UART guide.

One change that we have not discussed is the redefinition of the interrupts. In this guide, the configuration of the timer interrupt is left as Secure Group 0. However, in the second example it would be appropriate to have the timer interrupt set as Non-secure Group 1. The code for this has been included in the download. Noting the differences between the source files is something that you can do outside the scope of this guide.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6. Related information

Here are some resources related to material in this guide:

- Arm Community (ask development questions, and find articles and blogs on specific topics from Arm experts)
- Arm Cortex-A Series Programmer's Guide for Armv8-A (a chapter on security in general, and a section that covers using interrupts to switch between Secure and Non-secure worlds)
- Armv8-A Fundamentals guide (changing Exception levels and Security state)
- Armv8-A Learn the Architecture series of guides
- GICv3 and GICv4 Software Overview
- Scatter files

Here is some information about the various registers that are referred to in this guide:

- SCR_EL3, Secure Configuration Register
- SCTLR_EL1, System Control Register (EL1)
- ELR_EL3, Exception Link Register (EL3)
- SPSR_EL3, Saved Program Status Register (EL3)

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7. Next steps

This guide is the fourth in a series of four guides on the topic of building an embedded image. In this guide, we covered Exception levels and how to change them, and Security states and how to switch them.

In case you missed them, the previous guides in the series are:

- Building your first embedded image
- Retargeting output to UART
- Creating an event-driven embedded image

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential