ACLE Extensions for ARMv8-M

Version 2.0

Revision Information

The following revisions have been made to this User Guide.

Date	Issue	Confidentiality	Change
01 September 2016	0100	Non-Confidential	First release
28 February 2017	0200	Non-Confidential	Second release

Proprietary Notice

Confidentiality Status

This document is 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.

Web Address

http://www.arm.com

Contents

1 The ACLE Extensions for ARMv8-M

The ARM C Language Extensions (ACLE) for ARMv8-M enables the ARMv8-M Security Extension to build a Secure image, and to enable a Non-secure image to call a Secure image.

ACLE for ARMv8-M enables the ARMv8-M Security Extension you to write applications and middleware code that is portable across compilers, and across ARM architecture variants, while exploiting the unique features of the ARM architecture.

The ACLE specification specifies source language extensions and implementation choices that C/C++ compilers can implement to allow programmers to better exploit the ARM architecture.

The extensions include:

- Predefined macros that provide information about the functionality of the target architecture (for example, whether it has hardware floating-point).
- Intrinsic functions.
- Attributes that can be applied to functions, data, and other entities.

Some of the material, specifically the architecture and processor names, and the feature test macros, might also be applicable to assemblers and other tools. ACLE is not a Hardware Abstraction Layer (HAL), and does not specify a library component, but it might make it easier to write a HAL or other low-level library in C rather than assembler.

2 Building Secure and Non-secure images

ACLE provides tools allow you to build images that run in the Secure state of the ARMv8-M Security Extensions. You can also create an import library package that developers of Non-secure images must have for those images to call the Secure image.

For example, using amclang:

To build an image that runs in the Secure state you must include the <arm_cmse.h> header in your code, and compile using the -mcmse armclang command-line option. Doing this makes the following available:

- The Test Target, `TT`, instruction.
- `TT` instruction intrinsics.
- Non-secure function pointer intrinsics.
- The `__attribute__((cmse_nonsecure_call))` and `__attribute__((cmse_nonsecure_entry))` function attributes.

At startup, your Secure code must set up the *Secure Attribution Unit* (SAU) and call the Non-secure startup code.

2.1 Calling a Secure image from a Non-secure image using veneers

Calling a Secure image from a Non-secure image requires a transition from Non-secure to Secure state. A transition is initiated through Secure gateway veneers. Secure gateway veneers decouple the addresses from the rest of the Secure code.

An entry point in the Secure image, entryname, is identified with:

```
__acle_se_entryname:
entryname:
```

The calling sequence is as follows:

1. The Non-secure image uses the branch BL instruction to call the Secure gateway veneer for the required entry function in the Secure image:

   ```
   BL    entryname
   ```

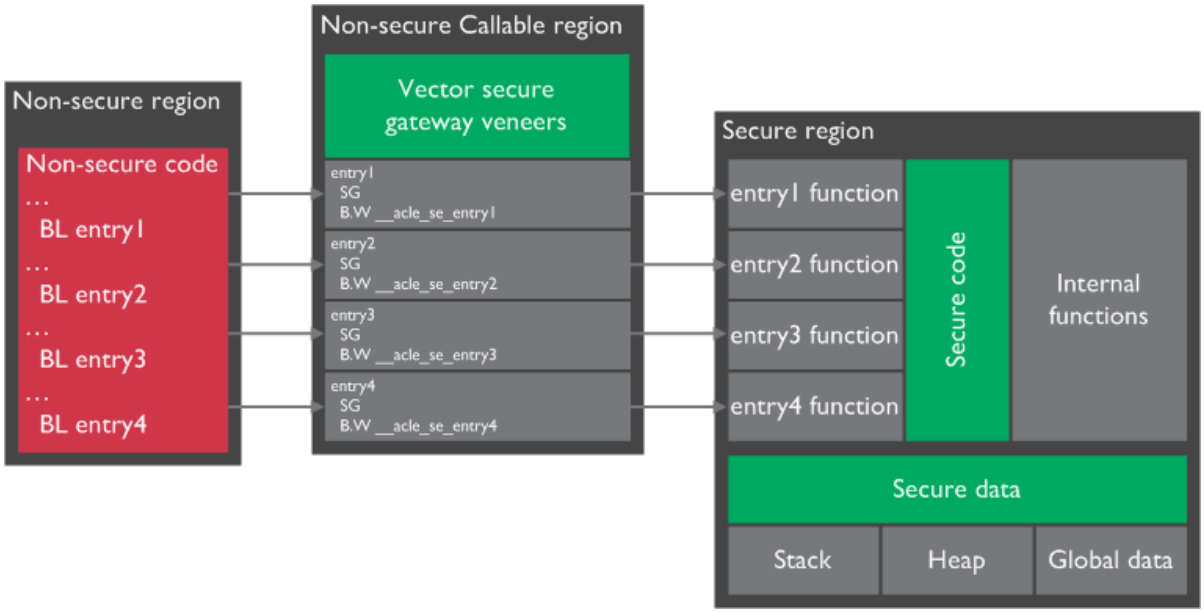
2. The Secure gateway veneer consists of the SG instruction and a call to the entry function in the Secure image using the B instruction:

   ```
   entryname
       SG
       B.W         __acle_se_entryname
   ```

3. The Secure image returns from the entry function using the BXNS instruction:

   ```
   BXNS  LR
   ```

The following figure is a graphical representation of the calling sequence, but for clarity, the return from the entry function is not shown:



2.2 Import library package

An import library package identifies the entry functions available in a Secure image. The import library package contains:

- An interface header file, for example `myinterface.h`. You manually create this file using any text editor.
- An import library, for example `importlib.o`. armlink generates this library during the link stage for a Secure image.

Note

You must have separate compile and link stages:

- To create an import library when building a Secure image.
- To use an import library when building a Non-secure image.

3 Building a Secure image using the ARMv8-M Security Extensions

When building a Secure image, you must also generate an import library that specifies the entry points to the Secure image. The import library is used when building a Non-secure image that calls the Secure image.

The following procedure is not a complete example, and assumes that your code sets up the Security Attribution Unit (SAU) and calls the Non-secure startup code.

Procedure

1. Create an interface header file, `myinterface_v1.h`, to be used by Non-secure code:

```
int entry1(int x);
int entry2(int x);
```

2. In the C program for your Secure code, `secure.c`, include the following:

```
#include <arm_cmse.h>
#include "myinterface_v1.h"

int func1(int x) { return x; }
int __attribute__((cmse_nonsecure_entry)) entry1(int x) { return func1(x) ; }
int __attribute__((cmse_nonsecure_entry)) entry2(int x) { return entry1(x); }

int main(void) { return 0; }
```

   In addition to the implementation of the two entry functions, the code defines the function func1() that can only be called by Secure code.

3. Create an object file using the armclang `-mcmse -mfloat-abi=soft` command-line options:

```
 $ armclang -c --target arm-arm-none-eabi -march=armv8-m.main -mcmse -mfpu=none
secure.c -o secure.o
```

4. To see the disassembly of the machine code that is generated by armclang, enter:

```
 $ armclang -c --target arm-arm-none-eabi -march=armv8-m.main -mcmse -mfloat-
abi=soft -S secure.c
```

   The disassembly is stored in the file `secure.s`, for example:

```
    .text
    ...
    .code 16
    .thumb_func
    ...
func1:
    .fnstart
    ...
    BX LR
    ...
```

```
__acle_se_entry1:
entry1:
    .fnstart
@ BB#0:
    save     {R7, LR}
    PUSH     {R7, LR}
    ...
    BL func1
    ...
    POP.W {R7, LR}
    ...
    BXNS LR
    ...
__acle_se_entry2:
entry2:
    .fnstart
@ BB#0:
    .save    {R7, LR}
    push     {R7, LR}
    ...
    BL entry1
    ...
    POP.W {R7, LR}
    BXNS LR
    ...
main:
    .fnstart
@ BB#0:
    ...
    MOVS R0, #0
    ...
    BX LR
    ...
```

The two symbols `__acle_se_entry_name` and `entry_name` indicate the start of an entry function to the linker. An entry function does not start with a Secure Gateway (SG) instruction.

5. You can control the placement of the section with the veneers using a scatter file and place it in your Non-secure Callable (NSC) region memory region. Create a scatter file containing the `Veneer$$CMSE` selector to place the entry function veneers, for example:

```
LOAD_REGION 0x0 0x3000
{
    EXEC_R 0x0
    {
        *(+RO,+RW,+ZI)
    }
    EXEC_NSCR 0x4000 0x1000
    {
        *(Veneer$$CMSE)
```

```
    }
    ARM_LIB_STACK 0x700000 EMPTY -0x10000
    {
    }
    ARM_LIB_HEAP  +0 EMPTY 0x10000
    {
    }
}
...
```

6. Using armclang you can link the object file using the armlink `--fpu softvfp` and `--import-cmse-lib-out` command-line options and the scatter file to create the Secure image:

   ```
   $ armlink secure.o -o secure.axf --cpu 8-M.Main --fpu SoftVFP --import-cmse-lib-out importlib_v1.o --scatter secure.scf
   ```

In addition to the final image, the link in this example also produces the import library, `importlib_v1.o`, for use when building a Non-secure image. Assuming that the section with veneers is placed at address `0x4000`, the import library consists of a relocatable file containing only a symbol table with the following entries:

Symbol type	Name	Address
STB_GLOBAL, SHN_ABS, STT_FUNC	entry1	0x4001
STB_GLOBAL, SHN_ABS, STT_FUNC	entry2	0x4009

Note

If you have an import library from a previous build of the Secure image, you can ensure that the addresses in the output import library do not change when producing a new version of the Secure image.

To do this, specify the `--import-cmse-lib-in` command-line option together with the `--import-cmse-lib-out` option. However, make sure that the input and output libraries have different names.

7. When you link the relocatable file corresponding to this assembly code into an image, the linker creates veneers in a section containing only entry veneers. To see the entry veneers generated by the linker, enter:

   ```
   $ fromelf --text -s -c secure.axf
   ```

   The following entry veneers are generated in the EXEC_NSCR execute-only (XO) region for this example:

```
    ...

    ** Section #3 'EXEC_NSCR' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR + SHF_ARM_NOREAD]
        Size   : 32 bytes (alignment 32)
        Address: 0x00004000

        $t
        entry1
```

```
            0x00004000:    E97FE97F    ....    SG        ; [0x3E08]
            0x00004004:    F7FCB85E    ..^.    B         __acle_se_entry1 ; 0xC4
        entry2
            0x00004008:    E97FE97F    ....    SG        ; [0x3E10]
            0x0000400c:    F7FCB86C    ..l.    B         __acle_se_entry2 ; 0xE8

    ...
```

The section with the veneers is aligned on a 32-byte boundary and padded to a 32-byte boundary.

If you do not use a scatter file, the entry veneers are placed in an ER_XO section as the first execution region, for example:

```
    ...

    ** Section #1 'ER_XO' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR +
SHF_ARM_NOREAD]
        Size   : 32 bytes (alignment 32)
        Address: 0x00008000

        $t
        entry1
            0x00008000:    E97FE97F    ....    SG        ; [0x7E08]
            0x00008004:    F000B85A    ..Z.    B.W       __acle_se_entry1 ;
0x80BC
        entry2
            0x00008008:    E97FE97F    ....    SG        ; [0x7E10]
            0x0000800c:    F000B868    ..h.    B.W       __acle_se_entry2 ;
0x80E0

    ...
```

Postrequisites

After you have built your Secure image:

1. Pre-load the Secure image onto your device.
2. Deliver your device with the pre-loaded image, together with the import library package, to a party who develops the Non-secure code for this device. The import library package contains:

- The interface header file, `myinterface_v1.h`.
- The import library, `importlib_v1.o`.

4 Building a Non-secure image that can call a Secure image

If you are building a Non-secure image that is to call a Secure image, you must obtain the import library package that was created for that Secure image.

The following procedure assumes that you have the import library package created using the ARMv8-M Security Extensions. The import library package identifies the entry points for the Secure image.

Procedure

1. In the C program for your Non-secure code, `nonsecure.c`, include the interface header file and use the entry functions as required, for example:

```
#include <stdio.h>
#include "myinterface_v1.h"

int main(void) {
    int val1, val2, x;

    val1 = entry1(x);
    val2 = entry2(x);

    if (val1 == val2) {
        printf("val2 is equal to val1\n");
    } else {
        printf("val2 is different from val1\n");
    }

    return 0;
}
```

2. Create an object file, `nonsecure.o`:

```
 $ armclang -c --target arm-arm-none-eabi -march=armv8-m.main nonsecure.c -o
nonsecure.o
```

3. Create a scatter file for the Non-secure image, but without the Non-secure Callable (NSC) memory region, for example:

```
LOAD_REGION 0x8000 0x3000
{
    ER 0x8000
    {
        *(+RO,+RW,+ZI)
    }
    ARM_LIB_STACK 0x800000 EMPTY -0x10000
    {
    }
    ARM_LIB_HEAP  +0 EMPTY 0x10000
```

```
        {
        }
    }
    ...
```

4. Link the object file using the import library, importlib_v1.o, and the scatter file to create the Non-secure image:

```
    $ armlink nonsecure.o importlib_v1.o -o nonsecure.axf --cpu=8-M.Main --
scatter nonsecure.scf
```

5 Building a Secure image using a previously generated import library

You can build a new version of a Secure image and use the same addresses for the entry points that were present in the previous version. You specify the import library that is generated for the previous version of the Secure image and generate another import library for the new Secure image.

The following procedure is not a complete example, and assumes that your code sets up the Secure Attribution Unit (SAU) and calls the Non-secure startup code. It assumes you have the import library package created using the ARMv8-M Security Extension. The import library package identifies the entry points for the Secure image.

Procedure

1. Create an interface header file, myinterface_v2.h, to be used by Non-secure code:

```
int entry1(int x);
int entry2(int x);
int entry3(int x);
int entry4(int x);
```

2. In the C program for your Secure code, secure.c, include the following:

```
#include <arm_cmse.h>
#include "myinterface_v2.h"

int func1(int x) { return x; }
int __attribute__((cmse_nonsecure_entry)) entry1(int x) { return func1(x) ; }
int __attribute__((cmse_nonsecure_entry)) entry2(int x) { return entry1(x); }
int __attribute__((cmse_nonsecure_entry)) entry3(int x) { return func1(x) +
entry1(x) ; }
int __attribute__((cmse_nonsecure_entry)) entry4(int x) { return entry1(x) *
entry2(x); }

int main(void) { return 0; }
```

   In addition to the implementation of the two entry functions, the code defines the function `func1()` that can only be called by Secure code.

3. Create an object file using the armclang -mcmse -mfloat-abi=soft command-line options:

```
 $ armclang -c --target arm-arm-none-eabi -march=armv8-m.main -mcmse -mfloat-
abi=soft secure.c -o secure.o
```

4. To see the disassembly of the machine code that is generated by armclang, enter:

```
 $ armclang -c --target arm-arm-none-eabi -march=armv8-m.main -mcmse -mfloat-
abi=soft -S secure.c
```

   The disassembly is stored in the file `secure.s`, for example:

```
    .text
    ...
```

```
    .code 16
    .thumb_func

    ...

func1:
    .fnstart
    ...
    BX LR

    ...

__acle_se_entry1:
entry1:
    .fnstart
@ BB#0:
    .save       {R7, LR}
    PUSH {R7, LR}
    ...
    BL func1
    POP.W {R7, LR}
    ...
    BXNS LR

    ...

__acle_se_entry4:
entry4:
    .fnstart
@ BB#0:
    .save       {R7, LR}
    PUSH {R7, LR}
    ...
    BL entry1
    ...
    POP.W {R7, LR}
    BXNS LR

    ...

main:
    .fnstart
@ BB#0:
    ...
    MOVS R0, #0
    ...
    BX LR

    ...
```

An entry function does not start with a Secure Gateway (SG) instruction. The two symbols __acle_se_entry_name and entry_name indicate the start of an entry function to the linker.

5. You can control the placement of the section with the veneers using a scatter file and place it in your NSC memory region. Create a scatter file containing the Veneer$$CMSE selector to place the entry function veneers, for example:

```
LOAD_REGION 0x0 0x3000
{
    EXEC_R 0x0
    {
        *(+RO,+RW,+ZI)
    }
    EXEC_NSCR 0x4000 0x1000
    {
        *(Veneer$$CMSE)
    }
    ARM_LIB_STACK 0x700000 EMPTY -0x10000
    {
    }
    ARM_LIB_HEAP  +0 EMPTY 0x10000
    {
    }
}
...
```

6. Using armclang, link the object file using the armlink --fpu softvfp, --import-cmse-lib-out, --import-cmse-lib-in command-line option and the preprocessed scatter file to create the Secure image:

```
$ armlink secure.o -o secure.axf --cpu 8-M.Main --fpu SoftVFP --import-cmse-lib-out importlib_v2.o --import-cmse-lib-in importlib_v1.o --scatter secure.scf
```

In addition to the final image, the link in this example also produces the import library, importlib_v2.o, for use when building a Non-secure image. Assuming that the section with veneers is at address 0x4000, the import library consists of a relocatable file containing only a symbol table with the following entries:

Symbol type	Name	Address
STB_GLOBAL, SHN_ABS, STT_FUNC	entry1	0x4001
STB_GLOBAL, SHN_ABS, STT_FUNC	entry2	0x4009
STB_GLOBAL, SHN_ABS, STT_FUNC	entry3	0x4021
STB_GLOBAL, SHN_ABS, STT_FUNC	entry4	0x4029

7. When you link the relocatable file corresponding to this assembly code into an image, the linker creates veneers in a section containing only entry veneers. To see the entry veneers generated by the linker, enter:

```
$ fromelf --text -s -c secure.axf
```

The following entry veneers are generated in the EXEC_NSCR execute-only (XO) region for this example:

```
    ...

    ** Section #3 'EXEC_NSCR' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR +
SHF_ARM_NOREAD]
        Size   : 64 bytes (alignment 32)
        Address: 0x00004000

        $t
        entry1
            0x00004000:    E97FE97F    ....    SG        ; [0x3E08]
            0x00004004:    F7FCB85E    ..^.    B         __acle_se_entry1 ; 0xc4
        entry2
            0x00004008:    E97FE97F    ....    SG        ; [0x3E10]
            0x0000400c:    F7FCB86C    ..l.    B         __acle_se_entry2 ; 0xe8

    ...

        entry3
            0x00004020:    E97FE97F    ....    SG        ; [0x3E28]
            0x00004024:    F7FCB872    ..r.    B         __acle_se_entry3 ; 0x10c
        entry4
            0x00004028:    E97FE97F    ....    SG        ; [0x3E30]
            0x0000402c:    F7FCB888    ....    B         __acle_se_entry4 ; 0x140
    ...
```

The section with the veneers is aligned on a 32-byte boundary and padded to a 32-byte boundary.

If you do not use a scatter file, the entry veneers are placed in an ER_XO section as the first execution region. The entry veneers for the existing entry points are placed in a CMSE veneer section. For example:

```
    ...

    ** Section #1 'ER_XO' (SHT_PROGBITS) [SHF_ALLOC + SHF_EXECINSTR +
SHF_ARM_NOREAD]
        Size   : 32 bytes (alignment 32)
        Address: 0x00008000

        $t
        entry3
            0x00008000:    E97FE97F    ....    SG        ; [0x7E08]
            0x00008004:    F000B87E    ..~.    B.W       __acle_se_entry3 ;
0x8104
        entry4
            0x00008008:    E97FE97F    ....    SG        ; [0x7E10]
            0x0000800c:    F000B894    ....    B.W       __acle_se_entry4 ;
0x8138
```

```
    ...

    ** Section #4 'ER$$Veneer$$CMSE_AT_0x00004000' (SHT_PROGBITS) [SHF_ALLOC +
SHF_EXECINSTR + SHF_ARM_NOREAD]
        Size   : 32 bytes (alignment 32)
        Address: 0x00004000

        $t
        entry1
            0x00004000:    E97FE97F    ....    SG        ; [0x3E08]
            0x00004004:    F004B85A    ..Z.    B.W       __acle_se_entry1 ;
0x80BC
        entry2
            0x00004008:    E97FE97F    ....    SG        ; [0x3E10]
            0x0000400c:    F004B868    ..h.    B.W       __acle_se_entry2 ;
0x80E0

    ...
```

Postrequisites

After you have built your updated Secure image:

1. Pre-load the updated Secure image onto your device.
2. Deliver your device with the pre-loaded image, together with the new import library package, to a party who develops the Non-secure code for this device. The import library package contains:

- The interface header file, `myinterface_v2.h`.
- The import library, `importlib_v2.o`.