ARMv8-M Tutorial: Cortex®-M23, Cortex-M33

Hands-On with ARM® Keil® MDK Toolkit

Winter 2019   Version 3.0

by Robert Boys, bob.boys@arm.com

Introduction:

This lab demonstrates operation of an ARMv8-M TrustZone® processor using Keil MDK. The ARMv8-M processor is run on the FVP (Fixed Virtual Platform built with ARM Fast Models) that is a component of Keil µVision. No external hardware is needed for this tutorial.

This tutorial simulates a Cortex-M23 processor Baseline processor with no RTOS. Examples using Cortex-M33 Mainline and with an RTOS are also available.

Software:

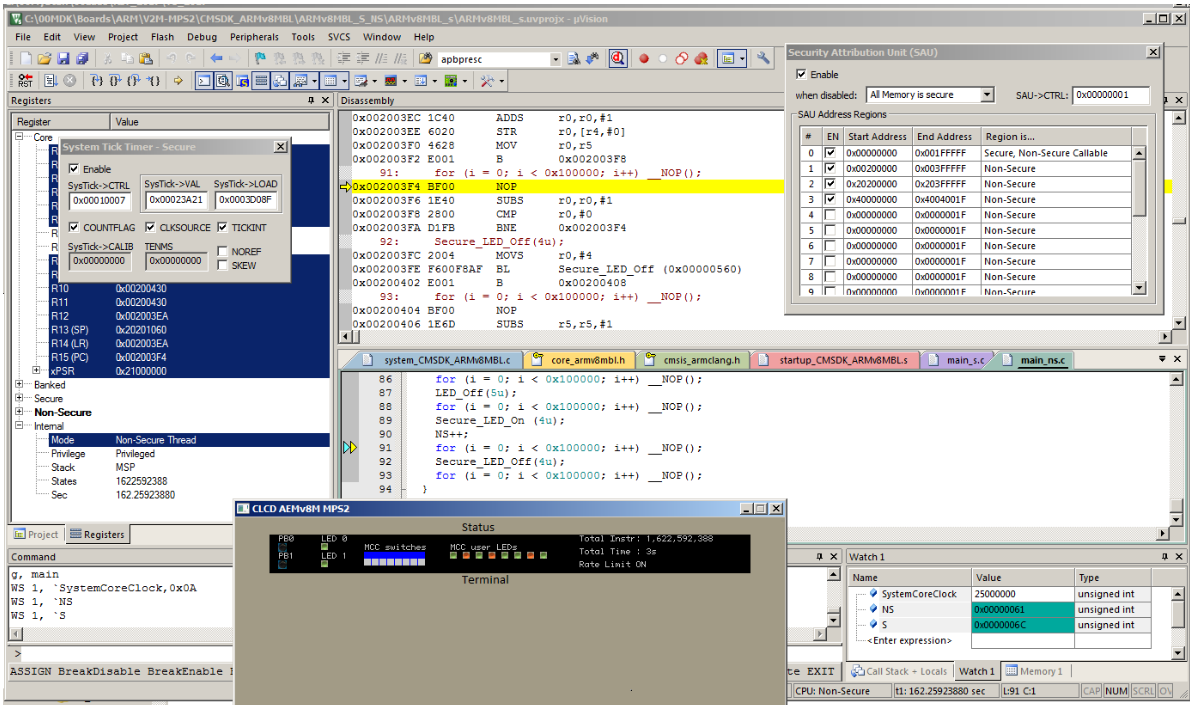
1. This lab uses Keil MDK 5.24a or later. You need an MDK Professional license.
2. The FVP simulator needs a 64 bit Windows PC. Keil MDK otherwise will work with either 64 or 32 bit PCs.
3. A Software Pack and the example must be downloaded from the Internet as described below.

Hardware:

No hardware is needed for this lab. If you require hardware, see the MPS2: www.keil.com/boards2/arm/v2m_mps2

ETM instruction trace and Serial Wire Viewer data trace are available when using the MPS2 board. These are ARM CoreSight™ components of ARM Cortex-M processors.

TIP: There are two states in an ARMv8-M processor: Secure and Non-Secure. These are indicated in project, filenames and µVision dialog boxes and windows as **_n** and **_ns** respectively in various names.
There are two main subgroups of ARMv8-M processors: Baseline and Mainline. They are Cortex-M23, Cortex-M33 and Cortex-M35P. They are identified with BL and ML in various project and file names.



C:\00MDK\Boards\ARM\V2M-MPS2\CMSDK_ARMv8MBL\ARMv8MBL_S_NS\ARMv8MBL_s\ARMv8MBL_s.uvprojx - µVision

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
apbpresc
Registers
Disassembly
Project
Registers
Command

System Tick Timer - Secure

☑ Enable

SysTick->CTRL	SysTick->VAL	SysTick->LOAD
0x00010007	0x00023A21	0x0003D08F

☑ COUNTFLAG ☑ CLKSOURCE ☑ TICKINT

SysTick->CALIB	TENMS	
0x00000000	0x00000000	□ NOREF □ SKEW

Register	Value
Core	
R10	0x00200430
R11	0x00200430
R12	0x002003EA
R13 (SP)	0x20201060
R14 (LR)	0x002003EA
R15 (PC)	0x002003F4
xPSR	0x21000000
Banked	
Secure	
Non-Secure	
Internal	
Mode	Non-Secure Thread
Privilege	Privileged
Stack	MSP
States	1622592388
Sec	162.25923880

```
0x002003EC 1C40      ADDS     r0,r0,#1
0x002003EE 6020      STR      r0,[r4,#0]
0x002003F0 4628      MOV      r0,r5
0x002003F2 E001      B        0x002003F8
    91:      for (i = 0; i < 0x100000; i++) __NOP();
0x002003F4 BF00      NOP
0x002003F6 1E40      SUBS     r0,r0,#1
0x002003F8 2800      CMP      r0,#0
0x002003FA D1FB      BNE      0x002003F4
    92:      Secure_LED_Off(4u);
0x002003FC 2004      MOVS     r0,#4
0x002003FE F600F8AF  BL       Secure_LED_Off (0x00000560)
0x00200402 E001      B        0x00200408
    93:      for (i = 0; i < 0x100000; i++) __NOP();
0x00200404 BF00      NOP
0x00200406 1E6D      SUBS     r5,r5,#1
```

Security Attribution Unit (SAU)

☑ Enable

when disabled: All Memory is secure   SAU->CTRL: 0x00000001

SAU Address Regions

#	EN	Start Address	End Address	Region is...
0	☑	0x00000000	0x001FFFFF	Secure, Non-Secure Callable
1	☑	0x00200000	0x003FFFFF	Non-Secure
2	☑	0x20200000	0x203FFFFF	Non-Secure
3	☑	0x40000000	0x4004001F	Non-Secure
4	□	0x00000000	0x0000001F	Non-Secure
5	□	0x00000000	0x0000001F	Non-Secure
6	□	0x00000000	0x0000001F	Non-Secure
7	□	0x00000000	0x0000001F	Non-Secure
8	□	0x00000000	0x0000001F	Non-Secure
9	□	0x00000000	0x0000001F	Non-Secure

```
    for (i = 0; i < 0x100000; i++) __NOP();
    LED_Off(5u);
    for (i = 0; i < 0x100000; i++) __NOP();
    Secure_LED_On (4u);
    NS++;
    for (i = 0; i < 0x100000; i++) __NOP();
    Secure_LED_Off(4u);
    for (i = 0; i < 0x100000; i++) __NOP();
  }
```

```
g, main
WS 1, `SystemCoreClock,0x0A
WS 1, `NS
WS 1, `S
>
ASSIGN BreakDisable BreakEnable 
```

CLCD AEMv8M MPS2

```
Total Instr: 1,622,592,388
Total Time : 3s
Rate Limit ON
```

Watch 1

Name	Value	Type
SystemCoreClock	25000000	unsigned int
NS	0x00000061	unsigned int
S	0x0000006C	unsigned int
<Enter expression>		

CPU: Non-Secure   t1: 162.25923880 sec   L:91 C:1

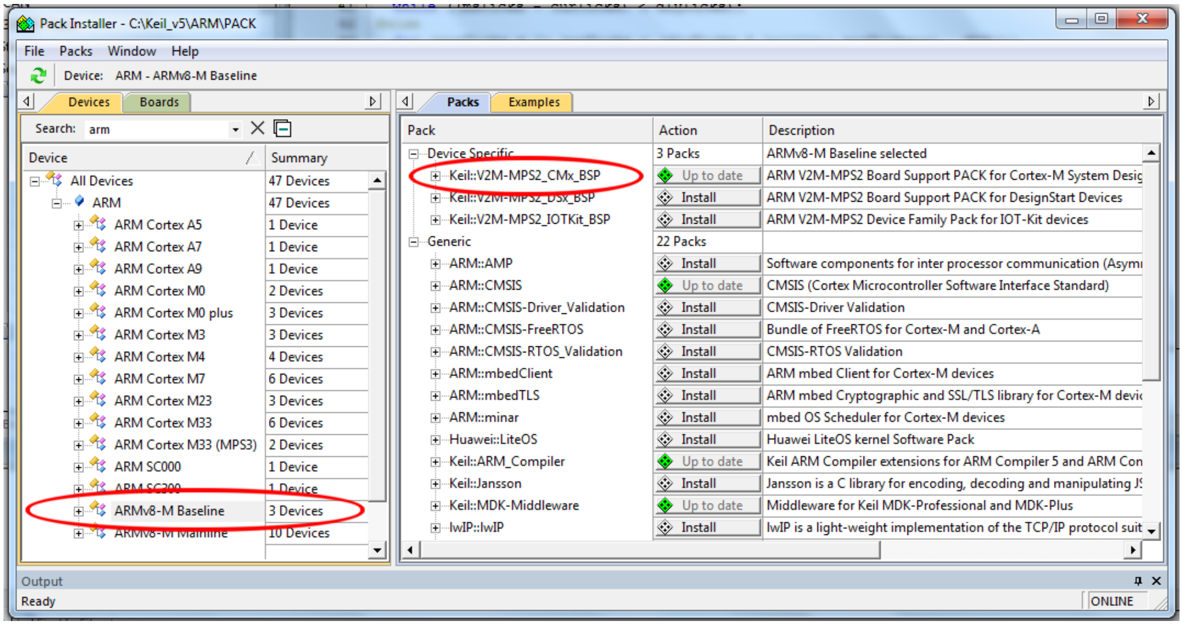
1) Install ARMv8-M BaseLine (BL) Software Pack: (PC must be connected to the Internet)

1. Start µVision® by clicking on its icon.
2. Open Pack Installer from inside µVision.
3. A window similar to the one below opens:
4. Select the Devices tab.
5. In the Search: box, enter ARM. Select ARMv8-M Baseline as shown below:
6. Select the Packs tab.
7. Click on the Install icon beside: Keil::V2M-MPS2_CMx_BSP as shown below: Install

TIP: If the icon says Up to Date, you do not need to install this Pack.

8. This Pack will download and install from the Internet. The Up to Date icon will now be visible as shown below.

TIP: What is selected on the left side in the Devices and Boards tabs and the Search box contents filters what is displayed on the right side in the Packs and Examples tabs.



Pack Installer - C:\Keil_v5\ARM\PACK

File Packs Window Help

Device: ARM - ARMv8-M Baseline

Devices | Boards

Search: arm

Device	Summary
All Devices	47 Devices
ARM	47 Devices
ARM Cortex A5	1 Device
ARM Cortex A7	1 Device
ARM Cortex A9	1 Device
ARM Cortex M0	2 Devices
ARM Cortex M0 plus	3 Devices
ARM Cortex M3	3 Devices
ARM Cortex M4	4 Devices
ARM Cortex M7	6 Devices
ARM Cortex M23	3 Devices
ARM Cortex M33	6 Devices
ARM Cortex M33 (MPS3)	2 Devices
ARM SC000	1 Device
ARM SC300	1 Device
ARMv8-M Baseline	3 Devices
ARMv8-M Mainline	10 Devices

Packs | Examples

Pack	Action	Description
Device Specific	3 Packs	ARMv8-M Baseline selected
Keil::V2M-MPS2_CMx_BSP	Up to date	ARM V2M-MPS2 Board Support PACK for Cortex-M System Desig
Keil::V2M-MPS2_DSx_BSP	Install	ARM V2M-MPS2 Board Support PACK for DesignStart Devices
Keil::V2M-MPS2_IOTKit_BSP	Install	ARM V2M-MPS2 Device Family Pack for IOT-Kit devices
Generic	22 Packs	
ARM::AMP	Install	Software components for inter processor communication (Asymı
ARM::CMSIS	Up to date	CMSIS (Cortex Microcontroller Software Interface Standard)
ARM::CMSIS-Driver_Validation	Install	CMSIS-Driver Validation
ARM::CMSIS-FreeRTOS	Install	Bundle of FreeRTOS for Cortex-M and Cortex-A
ARM::CMSIS-RTOS_Validation	Install	CMSIS-RTOS Validation
ARM::mbedClient	Install	ARM mbed Client for Cortex-M devices
ARM::mbedTLS	Install	ARM mbed Cryptographic and SSL/TLS library for Cortex-M devi
ARM::minar	Install	mbed OS Scheduler for Cortex-M devices
Huawei::LiteOS	Install	Huawei LiteOS kernel Software Pack
Keil::ARM_Compiler	Up to date	Keil ARM Compiler extensions for ARM Compiler 5 and ARM Con
Keil::Jansson	Install	Jansson is a C library for encoding, decoding and manipulating J
Keil::MDK-Middleware	Up to date	Middleware for Keil MDK-Professional and MDK-Plus
lwIP::lwIP	Install	lwIP is a light-weight implementation of the TCP/IP protocol suit

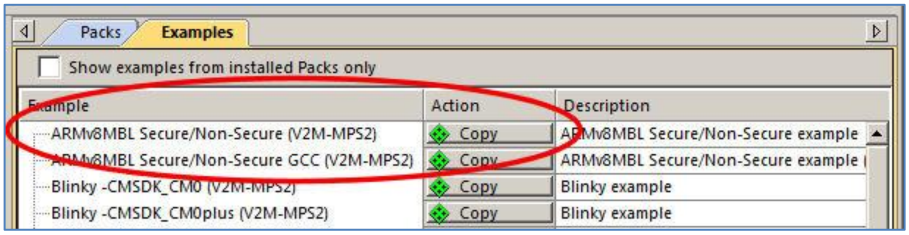
Output

Ready

ONLINE

2) Download the Example:

1. Select the Examples tab.
2. Select Copy Copy beside ARMv8MBL Secure/Non-Secure (v2M-MPS2) as shown here:

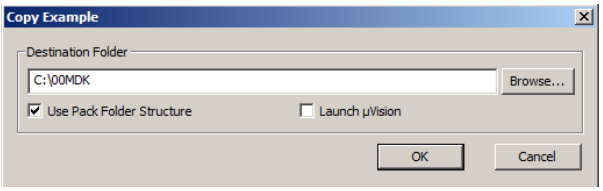


Packs | Examples

☐ Show examples from installed Packs only

Example	Action	Description
ARMv8MBL Secure/Non-Secure (V2M-MPS2)	Copy	ARMv8MBL Secure/Non-Secure example
ARMv8MBL Secure/Non-Secure GCC (V2M-MPS2)	Copy	ARMv8MBL Secure/Non-Secure example (
Blinky -CMSDK_CM0 (V2M-MPS2)	Copy	Blinky example
Blinky -CMSDK_CM0plus (V2M-MPS2)	Copy	Blinky example

3. The Copy Example window below opens up: Select Use Pack Folder Structure. Unselect Launch µVision:
4. Type in **C:\00MDK** as shown here:



Copy Example

Destination Folder: C:\00MDK

☑ Use Pack Folder Structure ☐ Launch µVision

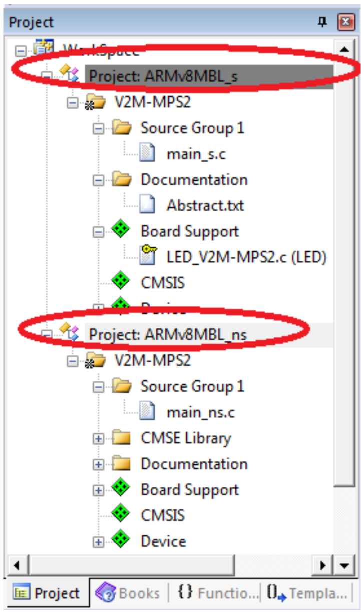
5. Click OK to copy the example file into C:\00MDK\Boards\ARM\V2M-MPS2\CMSDK_ARMv8MBL\ARMv8MBL_S_NS\
6. Pack Installer creates the appropriate subfolders.
7. Close the Pack Installer window.
8. At this point, you have everything you need to run the example and experiment with an ARMv8-M processor.

3) Open the example program:

1. Start µVision by clicking on its desktop icon if it is not already running.
2. Select Project/Open Project from the µVision main menu.
3. Navigate to: C:\00MDK\Boards\ARM\V2M-MPS2\CMSDK_ARMv8MBL\ ARMv8MBL_S_NS\
4. Open the µVision multi-project file: ARMv8MBL_s_ns.uvmpw
5. This project is a µVision multi-project workspace. One workspace is for the Secure program and the other is for Non-Secure. You can work on both at the same time. It is possible to deny µVision access to the Secure state.

µVision Multi- Project:

1. Open the Project window by clicking on its tab. This window opens:
2. Note there are two projects: ARMv8MBL_s and ARMv8MBL_ns.
3. The first one is Base Line (BL) Secure (**_s**) as shown to the right:
4. The second is Base Line (BL) Non-Secure (**_ns**).
5. In this window, the Secure project is active as shown by the gray background.
6. You select which project is active by right-clicking on it.


Project
WorkSpace
Project: ARMv8MBL_s
V2M-MPS2
Source Group 1
main_s.c
Documentation
Abstract.txt
Board Support
LED_V2M-MPS2.c (LED)
CMSIS
Device
Project: ARMv8MBL_ns
V2M-MPS2
Source Group 1
main_ns.c
CMSE Library
Documentation
Board Support
CMSIS
Device

Select the Non-Secure Project:

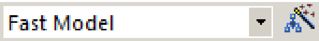
1. In the Project window, right-click on ARMv8MBL_ns and select Set as Active Project.

2. Change the Target Selector in the µVision upper task bar from V2M-MSP2 to Fast Model:

Fast Model

Select the Secure Project:

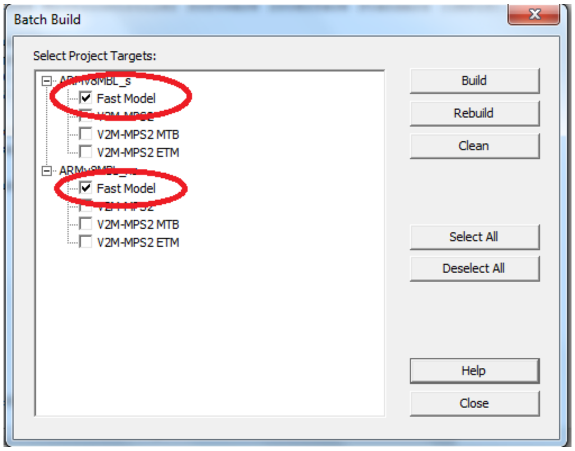
1. In the Project window, right-click on ARMv8MBL_s and select Set as Active Project.
2. Change the Target Selector from V2M-MSP2 to Fast Model:


Fast Model

3. Select File/Save All or click:

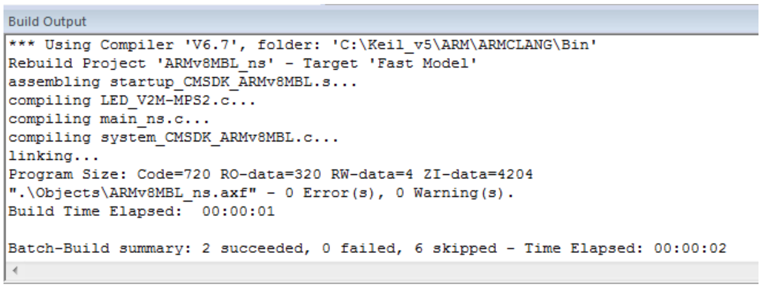
Build the two projects:

1. Click on the Batch Build icon: This window opens:
2. Unselect both instances of V2M-MPS2.
3. Select both instances of Fast Models as shown here:
4. Click Rebuild to compile both projects.
5. There will be no errors or warnings as shown below:


Batch Build

Select Project Targets:

ARMv8MBL_s
☑ Fast Model
☐ V2M-MPS2
☐ V2M-MPS2 MTB
☐ V2M-MPS2 ETM
ARMv8MBL_ns
☑ Fast Model
☐ V2M-MPS2
☐ V2M-MPS2 MTB
☐ V2M-MPS2 ETM


Build Output

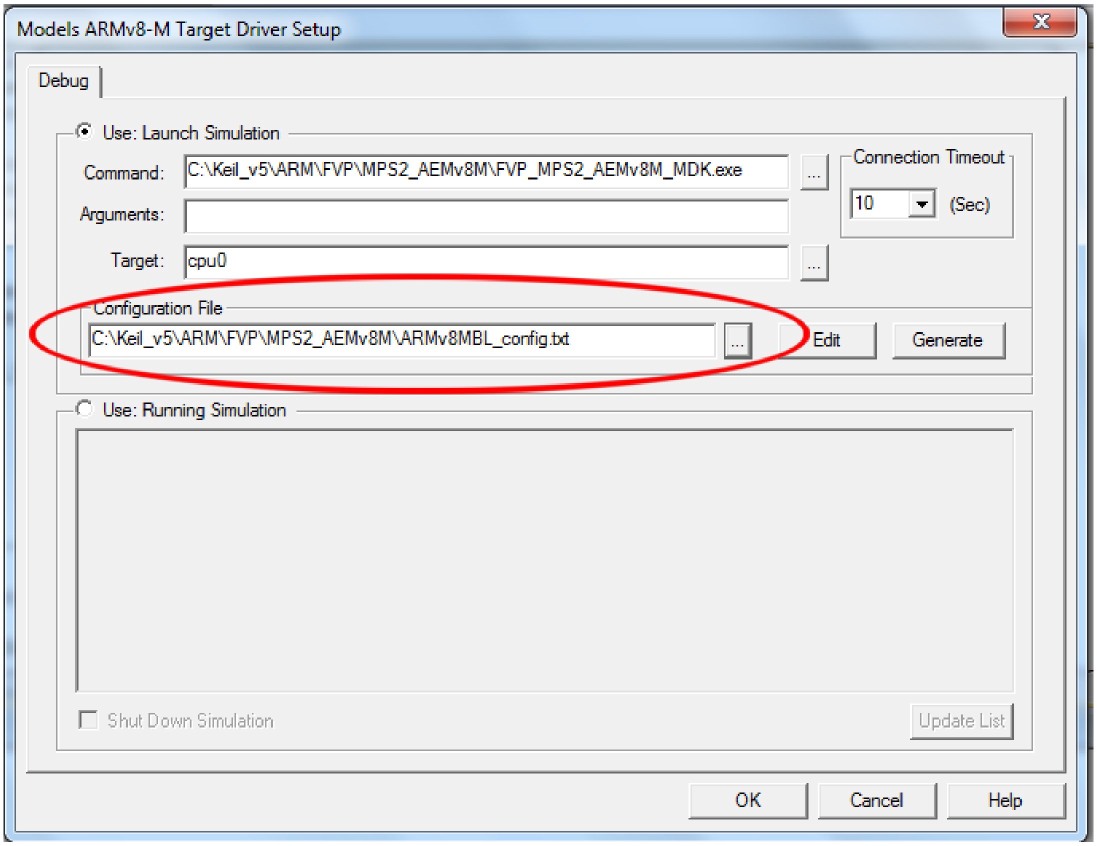
```
*** Using Compiler 'V6.7', folder: 'C:\Keil_v5\ARM\ARMCLANG\Bin'
Rebuild Project 'ARMv8MBL_ns' - Target 'Fast Model'
assembling startup_CMSDK_ARMv8MBL.s...
compiling LED_V2M-MPS2.c...
compiling main_ns.c...
compiling system_CMSDK_ARMv8MBL.c...
linking...
Program Size: Code=720 RO-data=320 RW-data=4 ZI-data=4204
".\Objects\ARMv8MBL_ns.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01

Batch-Build summary: 2 succeeded, 0 failed, 6 skipped - Time Elapsed: 00:00:02
```

For reference only: You probably do not have to perform the steps on this page.

If you are unable to enter Debug mode (next page) you may need to select the Configuration File.

4) Select FVP (Fixed Virtual Platform) .ini file:

1. Select Options for Target or ALT-F7.
2. Select the Debug tab.
3. Select the Settings: on the right side of this window. Settings
4. The Target Driver Setup window opens as shown below:
5. Using the Browse button for the Configuration File box, navigate to C:\Keil_v5\ARM\FVP\MPS2_AEMv8M.
6. Select the file ARMv8MBL_config.txt as shown.
7. Click OK twice to return to the main µVision window.



Models ARMv8-M Target Driver Setup

Debug

● Use: Launch Simulation

Command: C:\Keil_v5\ARM\FVP\MPS2_AEMv8M\FVP_MPS2_AEMv8M_MDK.exe

Arguments:

Target: cpu0

Connection Timeout: 10 (Sec)

Configuration File: C:\Keil_v5\ARM\FVP\MPS2_AEMv8M\ARMv8MBL_config.txt

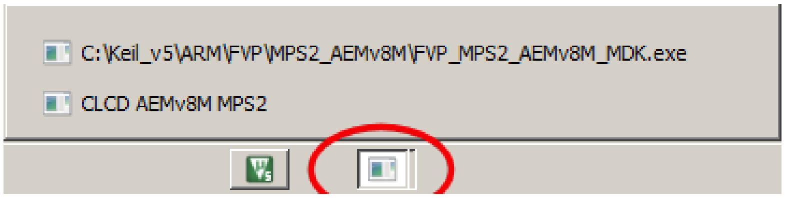
○ Use: Running Simulation

□ Shut Down Simulation

8. Select File/Save All or click .

5) Enter Debug Mode and RUN the Example program:

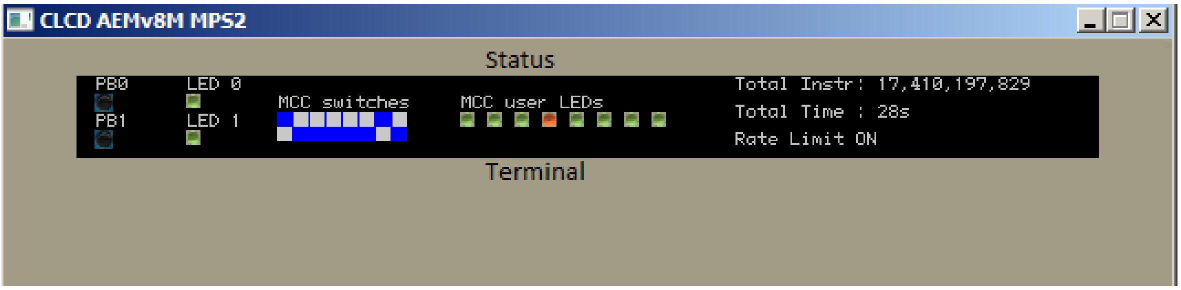
1. Enter Debug mode by clicking on the Debug icon.
2. Click on the RUN icon.
3. The CLCD window briefly appears and is then minimized.
4. This is the CLCD window and you can retrieve it by clicking on this icon in the Windows tool bar and then select CLCD AEMv8M MPS2 as shown here:
5. The CLCD window opens and the MCC view LEDs in the CLCD will blink.
6. You can click on the MCC switches on the blue boxes to change them.



C:\Keil_v5\ARM\FVP\MPS2_AEMv8M\FVP_MPS2_AEMv8M_MDK.exe

CLCD AEMv8M MPS2

<span style="color:red">**TIP:**</span> If you close the CLCD window by clicking on the **X** in the upper right corner, the Debug session will be terminated. If this happens, just start it again. It is easiest to let it minimize itself when you use µVision and select it when you need it.



CLCD AEMv8M MPS2

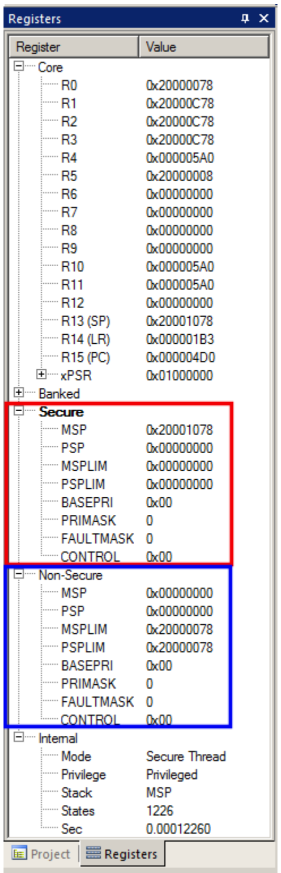
Status

```
PB0        LED 0                                          Total Instr: 17,410,197,829
                    MCC switches       MCC user LEDs      Total Time : 28s
PB1        LED 1                                          Rate Limit ON
```

Terminal

6) Determining if the CPU is in Secure or Non-Secure State:

1. Stop the program. It will probably stop in the Non-Secure state.
2. Open the Registers window by clicking on its tab. Registers
3. This window opens:



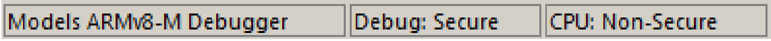
Registers

Register	Value
Core	
R0	0x20000078
R1	0x20000C78
R2	0x20000C78
R3	0x20000C78
R4	0x000005A0
R5	0x20000008
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x000005A0
R11	0x000005A0
R12	0x00000000
R13 (SP)	0x20001078
R14 (LR)	0x000001B3
R15 (PC)	0x000004D0
xPSR	0x01000000
Banked	
Secure	
MSP	0x20001078
PSP	0x00000000
MSPLIM	0x00000000
PSPLIM	0x00000000
BASEPRI	0x00
PRIMASK	0
FAULTMASK	0
CONTROL	0x00
Non-Secure	
MSP	0x00000000
PSP	0x00000000
MSPLIM	0x20000078
PSPLIM	0x20000078
BASEPRI	0x00
PRIMASK	0
FAULTMASK	0
CONTROL	0x00
Internal	
Mode	Secure Thread
Privilege	Privileged
Stack	MSP
States	1226
Sec	0.00012260

4. You can tell if the program is in Non-Secure or Secure state in this window.
5. In this window, it is in Secure state as **Secure** is bold.
6. This example spends most of its time in the while(1) loop in main_ns.c. Therefore, when you stop the program it will probably be in Non-Secure state. This is because of the large number of the __NOP intrinsic making a long timing delay in the for loop.
7. You can also see in the Internal header shown below, it is labelled Secure Thread.
8. On the next page, we will see how to go from Non-Secure to Secure state and back.

Task Bar:

9. At the bottom of the µVision window while in Debug state is a bar indicating the state access of the µVision debugger and the CPU as shown here:

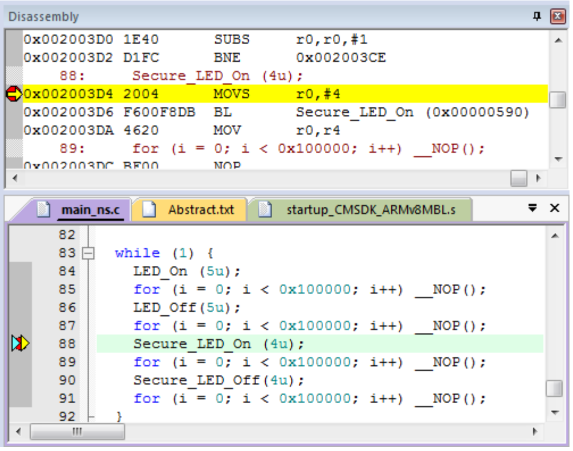


Models ARMv8-M Debugger | Debug: Secure | CPU: Non-Secure

TIP: It is possible to set the debugger so it can access either Secure and Non-Secure state or neither for security reasons. You might want to supply compiled code as in a library but not want a second party to debug it. In this case, you can allow debug access to only the non-secure state or bar access completely.

7) Run Program from Non-Secure state to Secure state:

1. The program can be running or halted at this point.
2. Set a breakpoint in main_ns.c at the function call Secure_LED_On near line 88 as shown below:



```
Disassembly
0x002003D0 1E40      SUBS     r0,r0,#1
0x002003D2 D1FC      BNE      0x002003CE
    88:      Secure_LED_On (4u);
0x002003D4 2004      MOVS     r0,#4
0x002003D6 F600F8DB  BL       Secure_LED_On (0x00000590)
0x002003DA 4620      MOV      r0,r4
    89:      for (i = 0; i < 0x100000; i++) __NOP();
0x002003DC BF00      NOP
```

```
82
83   while (1) {
84     LED_On (5u);
85     for (i = 0; i < 0x100000; i++) __NOP();
86     LED_Off(5u);
87     for (i = 0; i < 0x100000; i++) __NOP();
88     Secure_LED_On (4u);
89     for (i = 0; i < 0x100000; i++) __NOP();
90     Secure_LED_Off(4u);
91     for (i = 0; i < 0x100000; i++) __NOP();
92   }
```

3. Click RUN and the program stop as shown here:
4. Note the address in the Disassembly window (in this case) is 0x0020 03D4. This is in the non-secure memory as specified in the SAU.
5. The Non-Secure state is displayed in the Registers window and the Task bar as described in the previous page.
6. Click on the Disassembly window to bring it in focus.

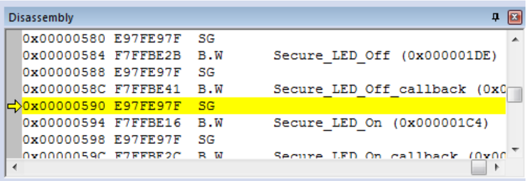
TIP: It is vital that the Disassembly window remains in focus. This ensures steps are by assembly instruction. Otherwise the steps will be by source code line and this is not what you want to see here. main_sc.c must be selected (underlined in its tab) to ensure the correct Disassembly window is in focus.

Enter Secure state:

1. Click STEP or F11 once to reach the BL instruction in Non-Secure memory at 0x0020 03D6.

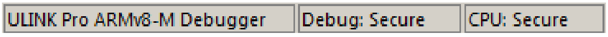
TIP: If the Disassembly window is not in focus and you step a source line, click RESET and run to the breakpoint. You can always exit and re-enter Debug mode to completely reset everything. Your breakpoint will be intact.

2. This is the Branch with Link to the Secure Gateway veneer located at 0x0000 0590.
3. Click STEP or F11 to execute this BL.
4. The program will jump to the veneer SG (Secure Gate) instruction at 0x0590 as shown here:
5. Note this is in the Secure memory area as specified by the SAU.



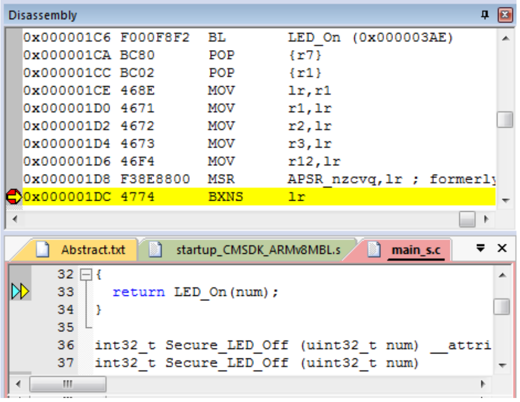
```
Disassembly
0x00000580 E97FE97F  SG
0x00000584 F7FFBE2B  B.W      Secure_LED_Off (0x000001DE)
0x00000588 E97FE97F  SG
0x0000058C F7FFBE41  B.W      Secure_LED_Off_callback (0x0
0x00000590 E97FE97F  SG
0x00000594 F7FFBE16  B.W      Secure_LED_On (0x000001C4)
0x00000598 E97FE97F  SG
0x0000059C F7FFBE2C  B.W      Secure_LED_On_callback (0x00
```

6. Click STEP or F11 to execute SG.
7. The CPU will now change to Secure state:



ULINK Pro ARMv8-M Debugger | Debug: Secure | CPU: Secure

8. Click STEP or F11 to execute B.W at 0x0594.
9. The program counter will now be at the beginning of the function Secure_LED_On.
10. Note the memory address is 0x01C4 which is in the Secure area as specified by the SAU.



```
Disassembly
0x000001C6 F000F8F2  BL       LED_On (0x000003AE)
0x000001CA BC80      POP      {r7}
0x000001CC BC02      POP      {r1}
0x000001CE 468E      MOV      lr,r1
0x000001D0 4671      MOV      r1,lr
0x000001D2 4672      MOV      r2,lr
0x000001D4 4673      MOV      r3,lr
0x000001D6 46F4      MOV      r12,lr
0x000001D8 F38E8800  MSR      APSR_nzcvq,lr ; formerly
0x000001DC 4774      BXNS     lr
```

```
{
  return LED_On(num);
}

int32_t Secure_LED_Off (uint32_t num) __attri
int32_t Secure_LED_Off (uint32_t num)
```

Exit Secure state and back to Non-secure:

1. Find the instruction BXNS LR near memory 0x01DC at the end of the function Secure_LED_On.
2. Set a breakpoint on this line. Click RUN . The program runs to this instruction. It does not execute it.
3. Unselect the breakpoint as we do not need it anymore.
4. BXNS is a new ARMv8-M instruction.
5. Click STEP or F11 to execute it.
6. The LED on the CLCD will change accordingly.
7. Note the CPU is in now the non-secure state.
8. The cycle is now complete. This is how a non-secure program calls a function in the secure state.
9. Select Debug/Breakpoints (or select Ctrl-B) and unselect the breakpoint you set in main_ns.c.

TIP: If anything is not as expected that might indicate an attack, a Secure Fault (Cortex-M33) or a Hard Fault (Cortex-M23) exception will result.

8) Viewing Secure and Non-Secure Variables:

We will now create two global variables to increment when in both the Secure and Non-Secure state. We will view these in the Watch 1 window. There are two global variables S and NS. They are incremented in either Secure or Non-Secure state.

1. Exit Debug mode.

Create the Secure variable:

1. In main_s.c, add these lines near line 9:

```
9    extern unsigned int S;
10  unsigned int S = 0;
```

2. In main_s.c, in the function Secure_LED_On which is near line 33, add this line this line just before the line return LED_On(num);

```
34    S++;
```

Create the Non-Secure variable:

1. In main_ns.c, add these two lines:

```
10    extern unsigned int NS;
11    unsigned int NS = 0;
```

2. In main_ns.c, just after the call to function Secure_LED_On (4u); which is near line 90, add this line:

```
90    NS++;
```

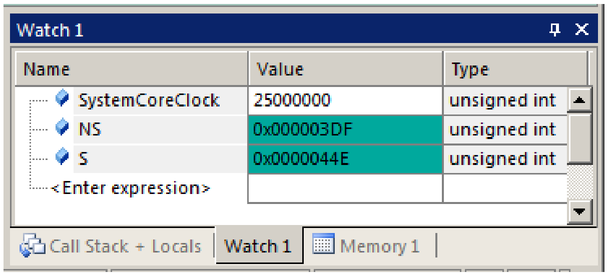
3. Select File/Save All or click:

Build the two projects:

4. Click on the Batch Build icon: The Batch Build window opens.
5. Click on Rebuild. Rebuild
6. Both project will build with no errors or warnings.
7. Enter Debug mode.

Run the Program and add variables to the Watch window:

1. Click Run . Note: Watch and Memory windows can have variables added while the program is running or not.
1. In main_ns.c, Right click on the variable NS and select Add 'NS to… and select Watch 1.
2. NS will be displayed in Watch 1 changing value.
3. In main_s.c, Right click on the variable S and select Add 'S to… and select Watch 1
2. Both variables are now displayed in Watch 1: →
3. This shows the different times when the CPU is running in either Secure or Non-Secure sate.


Watch 1

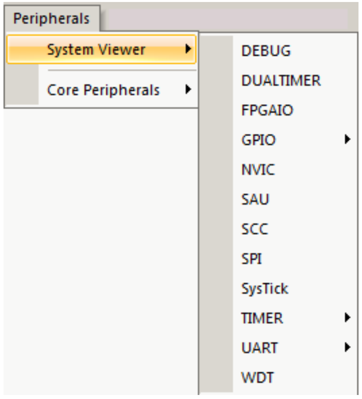
Name	Value	Type
SystemCoreClock	25000000	unsigned int
NS	0x000003DF	unsigned int
S	0x0000044E	unsigned int
<Enter expression>		

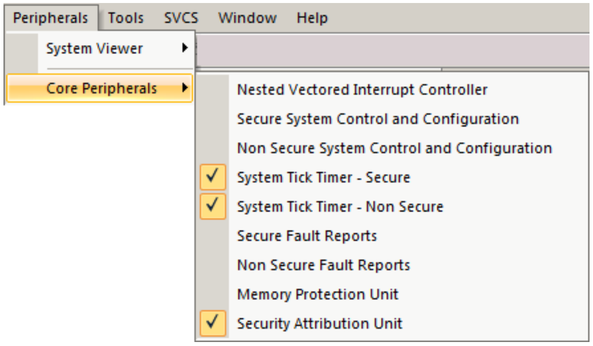
TIP: The debugger is in Secure state and can see all memory area. It can be restricted to see only the non-secure memory and registers.

9) Peripheral Windows:

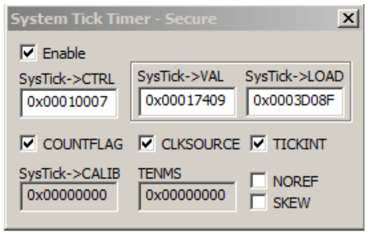
There are new registers in ARM V8-M processors and visibility if provided in µVision. These windows update in real-time as the program is running and in some cases, can be changed.

1. In µVision main menu, select Peripherals and note the two sections of peripherals available as shown here:
2. Select System Viewer and see the peripherals that are available:




Peripherals
Tools
SVCS
Window
Help
System Viewer
Core Peripherals
Nested Vectored Interrupt Controller
Secure System Control and Configuration
Non Secure System Control and Configuration
✓ System Tick Timer - Secure
✓ System Tick Timer - Non Secure
Secure Fault Reports
Non Secure Fault Reports
Memory Protection Unit
✓ Security Attribution Unit

Select Core Peripherals and open both System Tick Timer windows. If your program is running, you will see them these will update: No CPU cycles are stolen to do this.

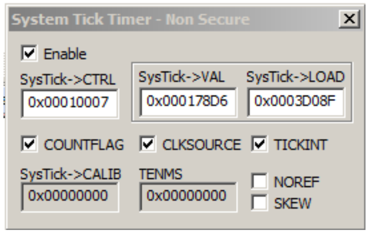

System Tick Timer - Secure

☑ Enable

SysTick->CTRL: 0x00010007 SysTick->VAL: 0x00017409 SysTick->LOAD: 0x0003D08F

☑ COUNTFLAG ☑ CLKSOURCE ☑ TICKINT

SysTick->CALIB: 0x00000000 TENMS: 0x00000000 □ NOREF □ SKEW


System Tick Timer - Non Secure

☑ Enable

SysTick->CTRL: 0x00010007 SysTick->VAL: 0x000178D6 SysTick->LOAD: 0x0003D08F

☑ COUNTFLAG ☑ CLKSOURCE ☑ TICKINT

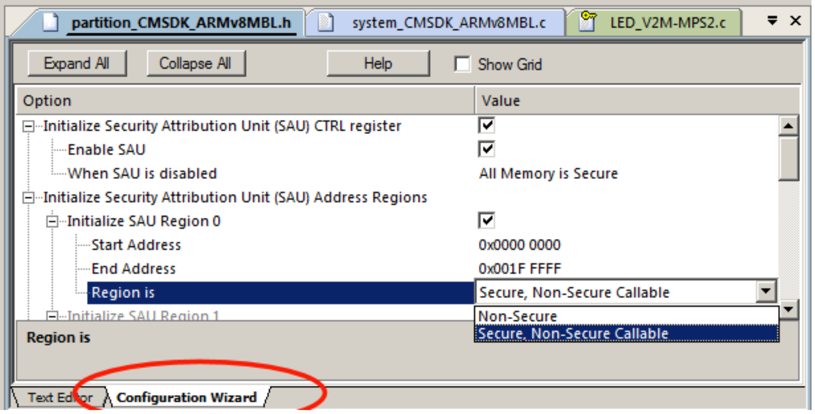
SysTick->CALIB: 0x00000000 TENMS: 0x00000000 □ NOREF □ SKEW

SAU: Security Attribution Unit:

1. The SAU defines memory that is Secure or Non-secure. This is defined in a new CMSIS file called partition_CMSDK_ARMv8MBL.h.
2. In the Project window under the Device heading, double click on it to open it.
3. Select the Configuration Wizard tab at the bottom of this file as shown below.
4. Note the various settings that can be selected. Do not make any changes.

SAU Peripheral:

1. Select Peripheral/Core Peripherals/Security Attribution Unit.
2. This window is shown below left:
3. Note the selections available. There is also a SAU window selectable under Peripherals/System Viewer. It is displayed in a different format.

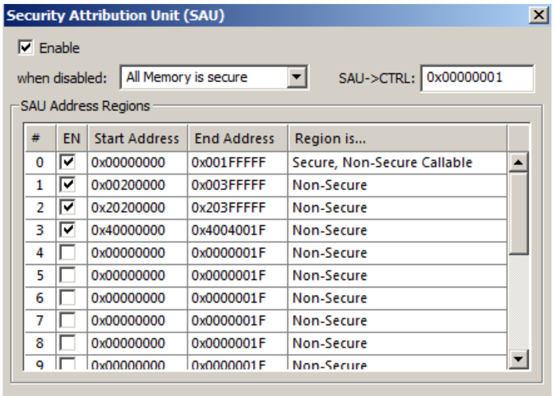


Option	Value
Initialize Security Attribution Unit (SAU) CTRL register	☑
Enable SAU	☑
When SAU is disabled	All Memory is Secure
Initialize Security Attribution Unit (SAU) Address Regions	
Initialize SAU Region 0	☑
Start Address	0x0000 0000
End Address	0x001F FFFF
Region is	Secure, Non-Secure Callable Non-Secure Secure, Non-Secure Callable
Initialize SAU Region 1	

Region is

TIP: You might have to stop the program to update the SAU window the first time.

This is the end of the ARMv8-M lab !


Security Attribution Unit (SAU)

☑ Enable

when disabled: All Memory is secure SAU->CTRL: 0x00000001

SAU Address Regions

#	EN	Start Address	End Address	Region is...
0	☑	0x00000000	0x001FFFFF	Secure, Non-Secure Callable
1	☑	0x00200000	0x003FFFFF	Non-Secure
2	☑	0x20200000	0x203FFFFF	Non-Secure
3	☑	0x40000000	0x4004001F	Non-Secure
4	□	0x00000000	0x0000001F	Non-Secure
5	□	0x00000000	0x0000001F	Non-Secure
6	□	0x00000000	0x0000001F	Non-Secure
7	□	0x00000000	0x0000001F	Non-Secure
8	□	0x00000000	0x0000001F	Non-Secure
9	□	0x00000000	0x0000001F	Non-Secure