TrueSTUDIO[®] Success

Working with bootloaders on Cortex-M devices





What is a bootloader?

General definition:

"A boot loader is a computer program that loads the main operating system or runtime environment for the computer after completion of the self-tests." - Wikipedia

In microcontroller land (ARM Cortex-M0/M3/M4/M7):

"A bootloader enriches the capabilities of the microcontroller and makes it a self-programmable device"

This is our definition!



Why use a bootloader?

Enables a device/product to upgrade itself in the field.

- Firmware is rarely bug free! Need a method to upgrade a product's firmware when defects are found.
- New requirements Need a method to upgrade a product's firmware due to new functionality.

A product recall might not be a feasible option!



Coverage

This document will cover the area of bootloaders from the perspective of Atollic TrueSTUDIO on ARM Cortex-M devices.

- Constructing and building the bootloader.
- Constructing and building the main application.
- Interaction between the bootloader and the main application.
- Use cases for debugging the above.



Coverage

It will <u>not</u> cover the actual self-update feature (downloading and flash reprogramming of the main application)

Many methods exists and are highly application specific!

- Update via USB, USART, CAN, SPI, ...
- Device capabilities.
- Device vendor support libraries.
- Etc.



Example hardware & code

STM32-F4-Discovery kit from STMicroelectronics

Download the example projects with ready-made code from TrueSTORE (inside TrueSTUDIO).

- File → New → Download new example project from TrueSTORE → STMicroelectronics → STM32F4-Discovery →
 - STM32F4_Discovery_Bootloader_APP
 - STM32F4_Discovery_Bootloader_BL

BL = Bootloader. APP = Application. Both must be downloaded.





Memory map

 Bootloader and application separated in flash (Important! Separated by flash pages!)

Our example:

- First 3 16K flash pages allocated to the bootloader.
 - 0x08000000 0x0800C000
 - Reset vector located at 0x0800000
- Rest of the flash allocated to the application
 - 0x0800C000 -





Constructing the bootloader

- Start with a project template generated by the project wizard. (STM32F4-Discovery board, code in flash memory.)
- Linker configuration file (stm32f4_flash.ld) will, by default, place the code at the start of flash at 0x08000000. This is what we want!

	🖹 stm32f4_flash.ld 🛛
Project Explorer X Sootloader X <	<pre>stm32f4_flash.ld X 11 ENTRY(Boot_Reset_Handler) 32 33 /* Highest address of the user mode stack */ 44_estack = 0x20020000; /* end of 128K RAM */ 35 36 /* Generate a link error if heap and stack don't fit into RAM */ 37 _Min_Heap_Size = 0; /* required amount of heap */ 38 _Min_Stack_Size = 0x400; /* required amount of stack */ 39 40 /* Specify the memory areas */ 41 MEMORY 42 { 43 FLASH (rx) : ORIGIN = 0x08000000, LENGTH = 1024K 47 RAM (xr) : ORIGIN = 0x20000000, LENGTH = 128K 45 MEMORY_B1 (rx) : ORIGIN = 0x20000000, LENGTH = 0K 46 CCMRAM (rx) : ORIGIN = 0x10000000, LENGTH = 64K 47 } 48 49 /* Define output sections */ 50 SECTIONS 51 { 52 /* The startup code goes first into FLASH */ 53 .isr_vector : 54 { 55 . = ALIGN(4); 56 KEEP(*(.isr_vector)) /* Startup code */ 57 . = ALIGN(4); 58 } >FLASH </pre>



Constructing the bootloader

To avoid confusing the debugger when debugging the bootloader and the application in the same debug session:

- Use different symbolic names for critical functions
- Simplifies breakpoint handling etc.

For instance the entry point for the bootloader will be the **Reset_Handler** function by default. You probably have another version with the same name in the application also!

- Rename Reset_Handler to Boot_Reset_Handler and update <u>all</u> references (7 in startup_stm32F40xx.s and 1 in stm32f4_flash.ld)!
- Rename main() to boot_main() and update all references!

```
S startup stm32f40xx.s 🔀
                                                                          stm32f4_flash.ld 🔀
 65 * @param None
                                                                           20 1
 66 * @retval : None
                                                                           29
 67 */
                                                                           30 /* Entry Point */
 68
                                                                           31 ENTRY(Boot_Reset_Handler)
 69 .section .text.Boot Reset Handler
                                                                           32
 70 .weak Boot Reset Handler
                                                                           33 /* Highest address of the user mode stack */
 71 .type Boot_Reset_Handler, %function
                                                                           34 estack = 0x20020000;
                                                                                                      /* end of 128K RAM */
 72 Boot_Reset_Handler:
                                                                           35
                         /* Atollic update: set stack pointer */
 73 ldr sp, =_estack
                                                                           36 /* Generate a link error if heap and stack don't fit into RAM */
 74
                                                                           37 Min Heap Size = 0;
                                                                                                   /* required amount of heap */
 75 /* Copy the data segment initializers from flash to SRAM */
                                                                           38 Min Stack Size = 0x400; /* required amount of stack */
  76 movs r1, #0
                                                                           39
 77 b LoopCopyDataInit
                                                                           40 /* Specify the memory areas */
 78
                                                                           41 MEMORY
 79 CopyDataInit:
                                                                          42 {
 80 ldr r3, = sidata
 81 ldr r3, [r3, r1]
                                                                                                                                            9
  82 str r3, [r0, r1]
  83 adds r1, r1, #4
```



A basic boot_main()

- Performs a firmware update if requested (not implemented in this example!).
- Sets up the environment for the application:
 - Locates and sets the application stack pointer address. (Stored at first entry in application vector table.)
 - Locates the application entry point. (Stored at second entry in application vector table.)
 - Configures the vector table offset register. (Exceptions/IRQ now finds its handlers here!)
 - Starts the application.



A basic boot_main()

```
💼 main.c 🔀
 37
 38
 39 /* Application start address */
 40 #define APPLICATION ADDRESS
                                  0x0800C000
 41
 42 typedef void (*pFunction)(void);
 43
 449 /**
 45 **-----
 46 ** Abstract: Bootloader
 48 */
49⊖ int boot_main(void)
 50 {
 51
 52
       pFunction appEntry;
 53
       uint32 t appStack;
 54
 55
       /* Check if firmware update required */
 56
       if(checkFirmwareUpdate()){
 57
 58
           /* Perform the update */
 59
           performFirmwareUpdate();
 60
61
       }
62
63
       /* Get the application stack pointer (First entry in the application vector table) */
64
65
       appStack = (uint32_t) *((__IO uint32_t*)APPLICATION_ADDRESS);
66
       /* Get the application entry point (Second entry in the application vector table) */
 67
       appEntry = (pFunction) *( IO uint32 t*) (APPLICATION ADDRESS + 4);
68
 69
       /* Reconfigure vector table offset register to match the application location */
 70
       SCB->VTOR = APPLICATION ADDRESS;
71
 72
       /* Set the application stack pointer */
 73
       __set_MSP(appStack);
 74
75
       /* Start the application */
 76
       appEntry();
 77
 78
       while(1);
 79
80 }
```



Constructing the application

- Start with a project template generated by the project wizard. (STM32F4-Discovery board, code in flash memory.)
- Linker configuration file (stm32f4_flash.ld) will, by default, place the code at the start of flash at 0x08000000 → 0x0800C000. We need to change this! You may also want to reduce LENGTH of FLASH 1024K- 3x16K = 976K





Constructing the application

NOTE!

In this example we have chosen to let the bootloader set up the basic environment (stack pointer, vector table, etc.) for the application.

Lookout for any application code that might circumvent this behavior!

For instance the "SystemInit" in this example:

#ifdef VECT_TAB_SRAM

SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */

#else

SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */

#endif



Constructing the application

Always a good idea to verify!

1989 Registers 🖾	🖾 📲 🖻 📩 🛃 🔪		SFRs 🕅			X ₁₆ X ₁₀ X ₂	<u>× ~ - e</u>
Name	Value	•	Register	Address	Value		
1010 r10	0x0		⊿				
1010 r11	0x0		▲ ñn SCB				=
1010 r12	0x0	_	▷ iiii CPUID	0xe000ed00	0x410fc241		
1010 sp	0x2001fff0		▶ 1919 ICSR	0xe000ed04	0x803		
1010 0101 lr	0x800c49b		▶ 1989 VTOR	0xe000ed08	0xc000		
1010 pc	0x800c4ba	=	▷ IIII AIRCR	0xe000ed0c	0xfa050000		
bioi xpsr	0x61000003		⊳ 1989 SCR	0xe000ed10	0x0		
1919 d0	0x0		⊳ 1989 CCR	0xe000ed14	0x200		
1101 d1	0x0		▷ 1999 SHPR1	0xe000ed18	0x0		
1111 d2	0x0		▷ 1111 SHPR2	0xe000ed1c	0x0		
1111 d3	0x0		▷ 1000 SHPR3	0xe000ed20	0x0		
1919 d4	0x0		▷ iiii SHCSR	0xe000ed24	0x0		
1889 d5	0x0		▷ 1989 CFSR	0xe000ed28	0x0		
1919 d6	0x0		▷ 1000 HFSR	0xe000ed2c	0x4000000		
1818 d7	0x0		▷ IIII DFSR	0xe000ed30	0x0		
1111 d8	0x0		▷ IIII MMFAR	0xe000ed34	0xe000edf8		
1888 d9	0x0		⊳ 3889 BFAR	0xe000ed38	0xe000edf8		
1818 d10	0x0		▷ 1989 AFSR	0xe000ed3c	0x0		
1919 d11	0x0		▷ iiii CPACR	0xe000ed88	0xf00000		
1919 d12	0x0		▷ ## SysTick				
1010 -14 -2	0.0		▷ ## NVIC				
Name : sp		r 	MSB 0 0 0 0 0 0 0 0	0000000	110000	0 0 0 0 0 0	0 0 0 LSB
Hex:0x2001fff0 Decimal:537001968 Octal:04000377760 Binary:1000000000001111111111110000 Default:0x2001fff0			Register: VTOR Address: 0xe000ed08 Value: 49152 Size: 32 Reset value: 0 Reset mask: 0xFFFFFFFF Access permission: RW				
•		+ +	Description: Vector Table Offset Register				



Putting it all together

Use cases during development

- Debugging the bootloader
- Debugging the application
- Debugging the bootloader & application
 - Application programmed by debugger
 - Application programmed by bootloader



Debugging the bootloader

- Create a debug configuration, **Download_and_debug_Bootloader**, for the Bootloader project.
- Edit the debug startup script and instruct the debugger to set a breakpoint at boot_main. (or Boot_Reset_Handler if debugging prior boot_main)
- Start the debugger!

Debug Configurations				
Create, manage, and run configurati	ons to the second se			
Ype filter text © C/C++ Application © C/C++ Application © C/C++ Postmoter Debugger © C/C++ Remote Application © Embedded C/C++ Application © Debug Bootloader ▶ Launch Group	Name: Download_and_debug Bootloader Image: Main Startup Scripts Source Common Target Hardware Initialization Script Target Software Startup Scripts Debug Analyze Unit Test Debug Analyze Unit Test Image: Image:			
Filter matched 7 of 7 items	Apply Revert			
?	Debug Close			



Debugging the bootloader

Debug - Bootloader/src/main.c - Atollic TrueSTUDIO® for ARM® Pro					
<u>File E</u> dit View <u>R</u> un Processor Expert <u>W</u> indow <u>H</u> elp					
- □ • → → → • • • • • • • • • • • • • • •	· Ⅲ ▼ 33 Et				
🏇 Debug 🛛 🧏 👻 🖓 🖓 🖓	🕬= Variables 🕴 💊 Breakpoints 📟 SFRs 🚻 P	egisters			
Download_and_debug Bootloader [Embedded C/C++ Application]	Name	Туре			
▲ i Bootloader.elf	(x)= appEntry	pFuncti			
Thread [1] <main> (Suspended : Breakpoint)</main>	(x)= appStack	uint32_t			
boot_main() at main.c:56 0x80004f2					
LoopFillZerobss() at startup_stm32f40xx.s:108 0x80004ae					
LoopFillZerobss() at startup_stm32f40xx.s:108 0x80004ae					
📓 gdb					
ST-LINK					
	4				
stm32f4_flash.ld					
47 **=					
48 */					
490 int boot_main(void)					
51					
52 pFunction appEntry;					
53 uint32_t appStack;					
54					
55 /* Check if firmware update required */					
57					
58 /* Perform the update */					
59 performFirmwareUpdate();					
60					
63 /* Get the application stack pointer (First entry in the applicat	ion vector table) */				
<pre>64 appStack = (uint32_t) *((IO uint32_t*)APPLICATION_ADDRESS);</pre>					
C.C.					



Debugging the bootloader

No source level debugging after branching off to the application (**appEntry()**)

More on that later!



Debugging the application

- Create a debug configuration, Download_and_debug_Application, for the Application project.
- The vector table offset register, stack pointer and program counter are setup just to be on the safe side! Might already be done by debugger or application startup code!
- Start the debugger!

	K
Image: Second	Name: Download_and_debug_Application Image: Main % Debugger (* Starup Scripts * Source) Common Target Hardware Initialization Script Target Software Starup Scripts Debug Debug Analyze Unit Test Image: Target Software Starup Scripts Debug Analyze Unit Test Image: Target Software Starup Scripts Debug Analyze Unit Test Image: Target Software Starup Scripts Debug Analyze Unit Test Image: Target Software Starup Scripts Image: Target Hardware Initialization Script Target Software Starup Scripts Image: Target Software Starup Scripts Image: Target Hardware Initialization Script Target Software Starup Scripts Image: Target Software Starup Scripts Image: Target Software Starup Script Target Software Starup Scripts Image: Target Software Starup Scripts Image: Target Software Starup Script Target Software Starup Scripts Image: Target Software Starup Scripts Image: Target Software Starup Script Target Software Starup Scripts Image: Target Software Starup Scripts Image: Target Software Starup Scripts Target Software Starup Scripts Image: Target Software Scripts Image: Target Software Scripts Target Software Scripts
Filter matched 8 of 8 items	Apply Reyer



Debugging the application

Elle Édit View Bun Processor Espert Window Help Image: Status	Debug - Application/src/main.c - Atollic TrueSTUDIO® for ARM® Pro				
Debug 33 Debug 33 Debug 34 Debug 33 Pread [1] < main> (Suspended : Breakpoint) Thread [1] < main> (Suspended : Breakpoin	<u>File Edit View Run Processor Expert Window H</u> elp				
Debug 3 Debug 3 Debug 3 Or Variables % Breakpoint % Expressions 33 Wit Variables % Breakpoint % Expressions 33 Wit Pariables % Breakpoint % Expression % Breakpoint % Breakpoint % Expression % Breakpoint % Breakpoint % Breakpoint % Expression % Breakpoint % Breakpo	🔛 🗟 🎄 韸 🕹 🕪 🗉 🔳 🦫 🧏 🤧 🕫 🗤 🔂 🔶 🕶 🖓 🖓				
Convolad_and_debug_Application [Embedded C/C++ Application] Application.eft Application.eft Add new expression Immed If an in (void) Add new expression Immed If an anin(void) Immed I = egi Add new expression Immed Immed I = new expression Immed Immed Immed I = new expression Immed Imm	🅸 Debug 🕴	💥 v 🗆 🛛	🕬= Variables 💁 Breakpoints 🍕	f Expressions 🖾	🐰 👪 Registers 🛋 Mo
<pre> Application.eff Application.eff Add new expression main(at main.c43.0x800.etba gdb ST-LINK main(x) Struck System.stm32t4xx. main.e Startup.stm32t40xxs</pre>	C Download_and_debug_Application [Embedded C/C++ Application]		Expression	Typ	be a second seco
	a 🎲 Application.elf		+ Add new expression		
gdb gdb gdb sT-LINK C main.c X C system_stm32Hxxc @ main.c S startup_stm32H0xxs T min.c X C system_stm32Hxxc @ startup_stm32H0xxs T min.c X C system_stm32Hxxc @ startup_stm32Hxxc S startup_startup_stm32Hxxc S startup_startup_stm32Hxxc S startup_stm32Hxxc S startup_startup_stm32Hxxc S startup_startup_startup_startup_stm32Hxxc S startup_startup_startup_stm32Hxxc S startup_startup_startup_stm32Hxxc S startup_startup_startup_startup_startup_startup_startup_stm32Hxxc S startup_star	Thread [1] <main> (Suspended : Breakpoint)</main>				
ST-LINK ST-	main() at main.c:43 0x800c4ba				
ST-Link mainc 13 f system_stm32f40x.c f mainc S startup_stm32f40x.s m	B ST INK				
2 main.c S © system_stm32M2x.c. @ main.c. S startup_stm32M2xx.s minut 37 ** Abstract: main program ** #* 38 ** Address Value 40 */ ** SCB 0 410 int main(void) 0xx000ed00 0x410fc241 0x400ed00 0x410fc241 42 int i = 0; *** *** 0xx000ed00 0x410fc241 0x000ed00 0x400fc241 44 */ *** *** 0xx000ed00 0x400fc241 0x0000ed00 0x400fc241 44 */ *** 0xx000ed00 0x400fc241 0x000ed00 0x400fc241 44 */ *** *** 0xx000ed00 0x400fc241 0x000ed00 0x400fc241 44 */ *** *** *** 0xx000ed10 0xd00 45 /*** *** *** 0xx000ed14 0x200 4 *** *** *** 0xx000ed14 0x200 50 *** **** **** 0xx000ed20 0x0 5	SI-LINK				
<pre> main.c S system_stm32f4xx.c @ main.c S startup_stm32f40x.s m m sequence startup_stm32f4xx.c @ main.c S startup_stm32f40x.s m sequence startup_stm32f4xx.c @ main.c S startup_stm32f40xx.s m m sequence startup_stm32f4xx.c @ main.c S startup_stm32f40xx.s m sequence startup_stm32f4xx.c @ main.c S startup_stm32f4xx.c @ main.c S startup_stm32f4xx.c @ main.c S startup_stm32f4xx.c @ main.c & ma</pre>					
<pre></pre>					
<pre> ************************************</pre>					
C main.c S system_stm32f4xx.c C main.c S startup_stm32f40xx.s 36 *** Abstract: main program *** Address Value 38 *** Address Value 49 */ *** Address Value 40 */ *** Address Value 40 */ *** Address Value 40 */ *** Address Value 41 int i = 0; *** **** Diff CPUID Doxe000ed04 Dox803 42 int i = 0; **** Diff CCR Doxe000ed14 Dox000 43 /*** if code has been located to RAM and interrupts are used. Diff CCR Doxe000ed14 Dox00 44 **** SCB->VTOR register. Diff CFSR Doxe000ed24 Dod 50 * SCB->VTOR register. ** E.g. SCB->VTOR register. Diff CFSR Doxe000ed28 Dod 53 */* ToDO - Add your application code here */ MSB 0 0 0 0 0 0 0 MSB 0 0 0 0 0 0 0 MSB 0 0 0 0 0 0 0 <td></td> <td></td> <td></td> <td></td> <td></td>					
Image: Control in the system_straight in the system_in					
36 *** 37 ** Abstract: main program 38 *** 39 *** 40 */ 41 int main(void) 42 *** 43 int i = 0; 44 /*** *** *** 44 /*** *** *** <tr< th=""><th>C main.c ⋈ C system_stm32f4xx.c C main.c S startup_stm32f40xx.s</th><th></th><th>SFRs 🛛</th><th></th><th></th></tr<>	C main.c ⋈ C system_stm32f4xx.c C main.c S startup_stm32f40xx.s		SFRs 🛛		
38 **	36 ** 37 ** Abstract: main program	*	Register	Address	Value
39 **	38 **		⊿ 👬 SCB		
410 int main(void) 42 { int i = 0; 44 int i = 0; 44 /** 45 /** 46 int i = 0; 47 * IMPORTANT NOTE! 48 * if code has been located to RAM and interrupts are used. 49 * Otherwise the interrupt table located in flash will be used. 50 * See also the <system_*.c> file and how the SystemInit() function updates 51 * SCB->VTOR register. 52 /* TODO - Add your application code here */</system_*.c>	39 **=		⊳ 8889 CPUID	0xe000ed00	0x410fc241
42 int i = 0; > 000 0000000000000000000000000000000	40 17 410 int main(void)		⊳ lilii ICSR	0xe000ed04	0x803
44 44 44 45 7** 46 * IMPORTANT NOTE! * The symbol VECT_TAB_SRAM needs to be defined when building the project * if code has been located to RAM and interrupts are used. * 0 therwise the interrupt table located in flash will be used. * ScB ->VTOR register. * E.g. SCB->VTOR register. * E.g. SCB->VTOR = 0x20000000; */ * * TODO - Add your application code here */ * * * * TODO - Add your application code here */ *	42 {		▷ iiii VTOR	0xe000ed08	0xc000
44 > *** b ****	♦ 43 int i = 0;	D NNN AIRCR	0xe000ed0c	0xfa050000	
46 * IMPORTANT NOTE! 47 * The symbol VECT_TAB_SRAM needs to be defined when building the project 48 * if code has been located to RAM and interrupts are used. 49 * Otherwise the interrupt table located in flash will be used. 50 * See also the <system_*.c> file and how the SystemInit() function updates 51 * SCB->VTOR register. 52 * E.g. 54 * 55 /* TODO - Add your application code here */</system_*.c>	44	⊳ bibi SCR	0xe000ed10	0x0	
47 * The symbol VECT_TAB_SRAM needs to be defined when building the project > 000 0xe000ed18 0x00 48 * if code has been located to RAM and interrupts are used. > 000 > 000 0xe000ed120 0x00 49 * Otherwise the interrupt table located in flash will be used. > 000 > 000 > 000 > 000 50 * Sc8->VTOR register. > 0xe000ed20 0x00 > 000 > 000 > 000 51 * SC8->VTOR register. > 0xe000ed28 0x00 > 000 > 000 > 000 > 000 53 */ * TODO - Add your application code here */ MSB 0 0 0 0 0 0 0 0 0 0 Register: VTOR Address: 0xe000ed08		▷ bibi CCR	0xe000ed14	0x200	
48 * if code has been located to RAM and interrupts are used. > iiii SHPR2 0xe000edic 0x0 49 * Otherwise the interrupt table located in flash will be used. > iiiii SHPR3 0xe000ed20 0x0 50 * Sce also the <system *.c=""> file and how the SystemInit() function updates > iiiii SHCSR 0xe000ed24 0x0 51 * SCB->VTOR register. > iiiii SHCSR 0xe000ed28 0x0 52 * E.g. SCB->VTOR = 0x20000000; > iiiii CFSR 0xe000ed28 0x0 53 */ MSB 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</system>	47 * The symbol VECT TAB SRAM needs to be defined when building the project	=		0xe000ed18	0x0
49 * Otherwise the interrupt table located in flash will be used. 50 * See also the <system_*.c> file and how the SystemInit() function updates 51 * SCB->VTOR register. 52 * E.g. SCB->VTOR = 0x20000000; 53 */ 54 MSB 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</system_*.c>	48 * if code has been located to RAM and interrupts are used.		⊳ öiöi SHPR2	0xe000ed1c	0x0
50 * See also the <system_*.c> file and how the SystemInit() function updates 51 * SCB->VTOR register. 52 * E.g. SCB->VTOR = 0x20000000; 53 */ 54 MSB 0 0 0 0 0 0 0 0 0 0 0 0 0 55 /* TOD0 - Add your application code here */ 56 Register: VTOR Address: 0xe000ed08</system_*.c>	49 * Otherwise the interrupt table located in flash will be used.		⊳ iiii SHPR3	0xe000ed20	0x0
S1 - SLD->VTOR register. 52 * E.g. S3 */ 54 MSB 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	50 * See also the <system_*.c> file and how the SystemInit() function updates</system_*.c>		⊳ NNN SHCSR	0xe000ed24	0x0
53 */ 54 54 55 /* TODO - Add your application code here */ 56 /* TODO - Add your application code here */	51 * SCB-SVIOR register. 52 * E.g. SCB-SVIOR = $0x20000000$		▷ iiiii CFSR	0xe000ed28	0x0
255 /* TOD0 - Add your application code here */ Register: VTOR 56 Address: 0xe000ed08	53 */ 54			MSB 0	0000000
56 Address: UxeU00edU8	55 /* TODO - Add your application code here */		Register: VTOR		
57 /* Initialize LEDs */ Value: 49152	56 57 /* Initialize LEDs */	-	Value: Va		



Debugging the application and bootloader

In order to do source-level debugging through both the bootloader and the application project "at the same time" some configuration is needed.

The debugger needs to have information regarding both of the ELF files!



Both the bootloader and application binary are to be programmed during a debug launch. Source-level debugging should work after the "jump".



Create a third debug configuration connected to the Bootloader project.

Debug Configurations			×
Create, manage, and run configurations			Ś
Yume Yume type filter text € C/C++ Application € C/C++ Attach to Application € C/C++ Postmortem Debugger € C/C++ Remote Application	Name: Download_and_debug_	Bootloader_and_Application Startup Scripts 1 Source Common Variables	Browse
 Embedded //C++ Application Download_and_debug Bootloader Download_and_debug_Application Download_and_debug_Bootloader_and_Application Launch Group 	<u>P</u> roject: Bootloader		<u>B</u> rowse
	Build (if required) before laune Build configuration:	Debug	
	 Enable auto build Use workspace settings 	Select configuration using 'C/C++ Applicat Disable auto build <u>Configure Workspace Settings</u> .	ion'
Filter matched 9 of 9 items		Apply	Revert
?		Debug	Close



Edit the debugger startup script and instruct the debugger to perform the following:

- Program the bootloader binary.
- Program the application binary.
- Since the application binary in this case was programmed last we need to readd the symbolic information for the bootloader binary.

The debugger is now aware of both the binaries!

- Reset target.
- Set breakpoint at bootloader entry and start execution.





Only the bootloader is to be programmed during a debug launch. The application is programmed by the bootloader (IAP). Source-level debugging should work after the "jump".



Create a fourth debug configuration connected to the Bootloader project. Use the "Download_and_debug Bootloader" configuration as template.

Debug Configurations		
Create, manage, and run configurations		
Image: Second Secon	Name: Download_and_debug_Boot Main	loader_debug_ApplicationIAP up Scripts 15/2 Source Qommon Yariables Search Project Browse Browse
 Download_and_debug_Bootloader_debug_ApplicationIAP Launch Group Filter matched 10 of 10 items 	Build configuration:	© Debug
	Use workspace settings	Configure Workspace Settings Apply Reyert
?		Debug Close



Edit the debugger startup script with the following change:

• Add symbolic/debug information from the corresponding application's ELF file.



More information:

www.atollic.com

EUROPE & WORLDWIDE

Atollic AB Science Park Gjuterigatan 7 SE-553 18 Jönköping Sweden +46 36 19 60 50

info@atollic.com

USA & AMERICAS

Atollic Inc.

241 Boston Post Road West Marlborough, Massachusetts 01752 +1 (617) 674-2655

sales.usa@atollic.com