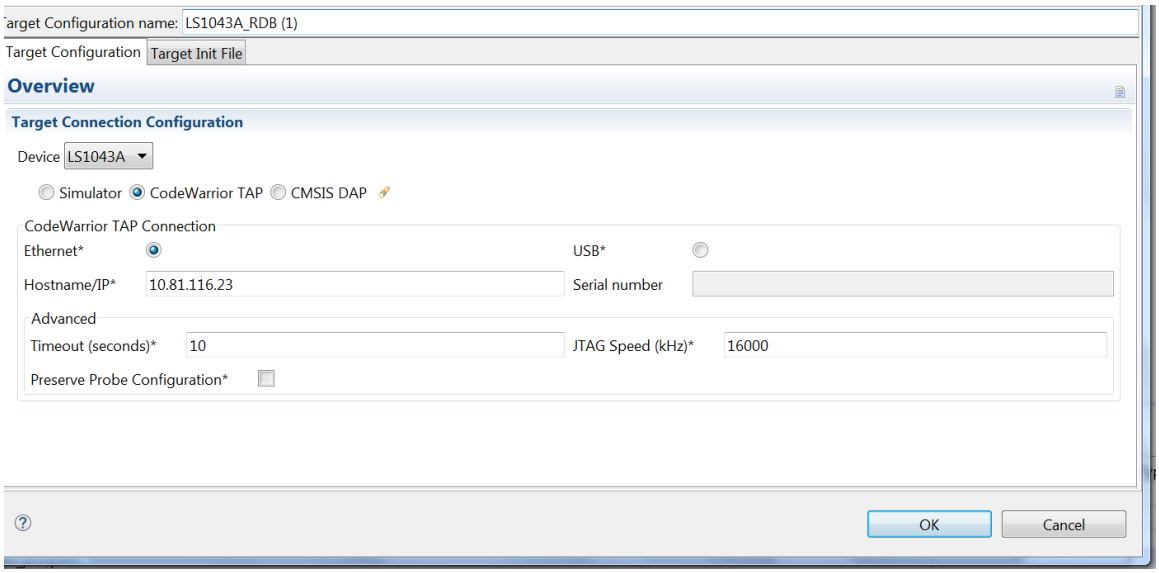
Use CodeWarrior for ARMv8 to Debug U-boot and Linux Kernel and Bring up Bare Boards

This document uses LS1043ARDB as an example to introduce how to use CodeWarrior for ARMv8 to connect to the bare board to do flash programming, how to use attaching method to debug u-boot and Linux Kernel on QorIQ LS ARM 64 bit target boards.

In the document some new features of CodeWarrior for ARMv8 are used, so it is needed to download the latest CodeWarrior Networked Application Suite v2016.01 and install CodeWarrior for ARMv8.

1. CodeWarrior Connects to the Bare Board to do Flash Programing

Before flash programing with CodeWarrior, the user needs to create a “Hello World” ARMv8 Bare Board project and use it to connect to the target board. Please create the project from File->New->ARMv8 Stationery, build this project, and configure target launch connection from Run->Debug Configurations->GDB Hardware Debugging-><project>->Debugger->Configure target connection, duplicate LS1043A_RDB target configuration and modify it as the following.

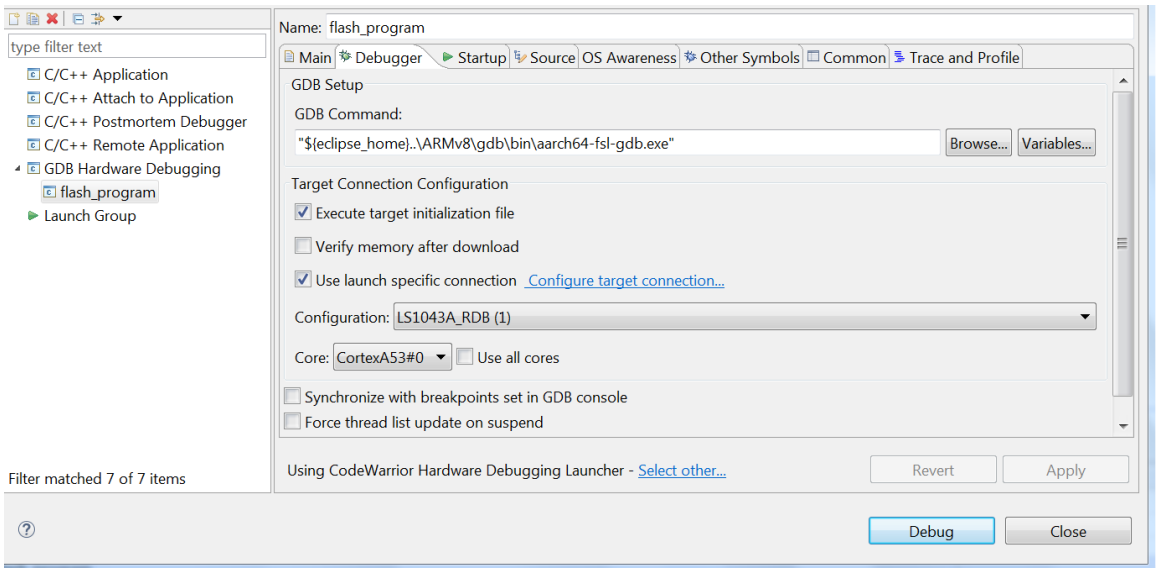


During the target board bringing up stage, the IFC flash is empty and there is no valid RCW on the target, the user could use CodeWarrior to modify the RCW source to hard-coded RCW, please modify the variable “useSafeRCW” as True in the target Initialization file, the default is false. Please refer to the follow.

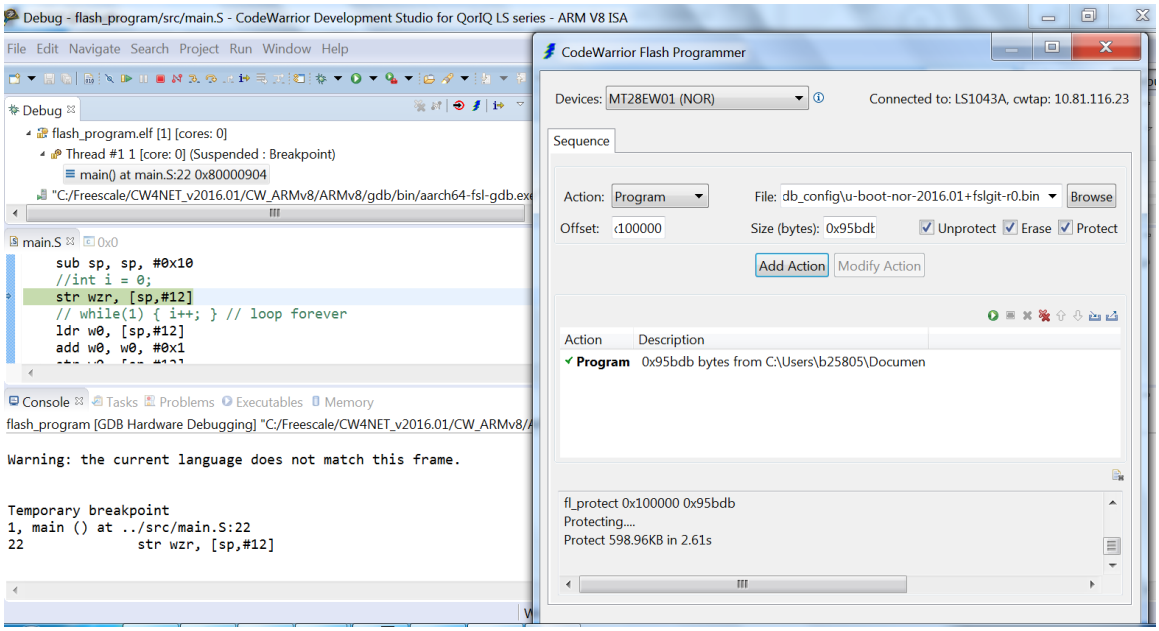
 # Uncomment to enable RCW override
useSafeRCW = True
Reset(useSafeRCW)

If the DDR or IFC flash on the custom board is different from the demo board, it is also needed to modify the DDR or IFC initialization section in the target initialization file. Users could use QCVS tool to assist them to decide DDR controller parameters according to the custom board.

After configure target connection, please check “Use launch specific connection” and set the connection as LS1043A_RDB(1) in the Debugger panel.



After CodeWarrior connects to the target board and enters into debug mode, please click the flash programmer icon to configure the task as the following to program u-boot to NOR Flash. Configure u-boot image offset on the NOR Flash, add and execution actions.



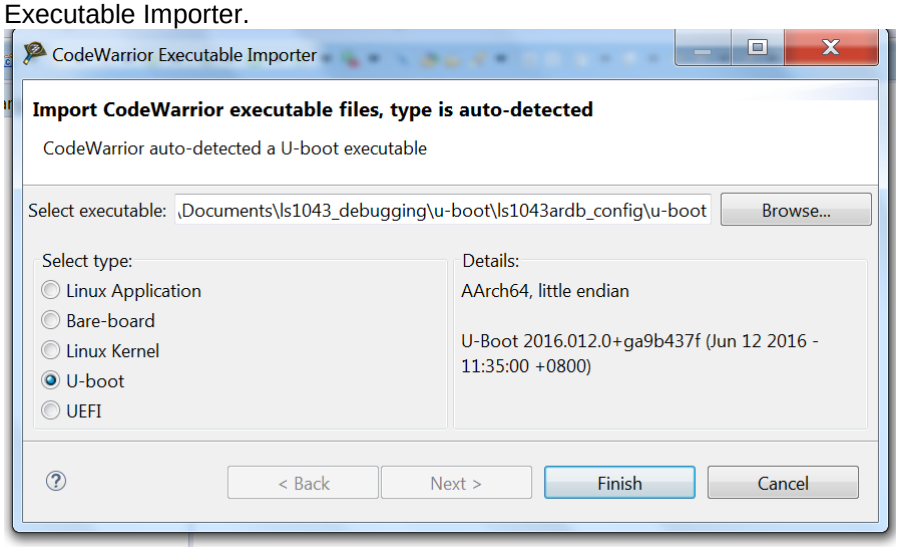
2. Debug u-boot with CodeWarrior for ARMv8

Build u-boot in SDK 2.0 Yocto environment with the following commands and copy the source code folder <install_folder>/ QorIQ-SDK-V2.0-20160527-yocto/build_ls1043ardb/tmp/work/ls1043ardb-fsl-linux/u-boot-qoriq/2016.01+fslgit-r0/git to a new location prepared for debugging, if CodeWarrior and SDK are installed in different host machines.

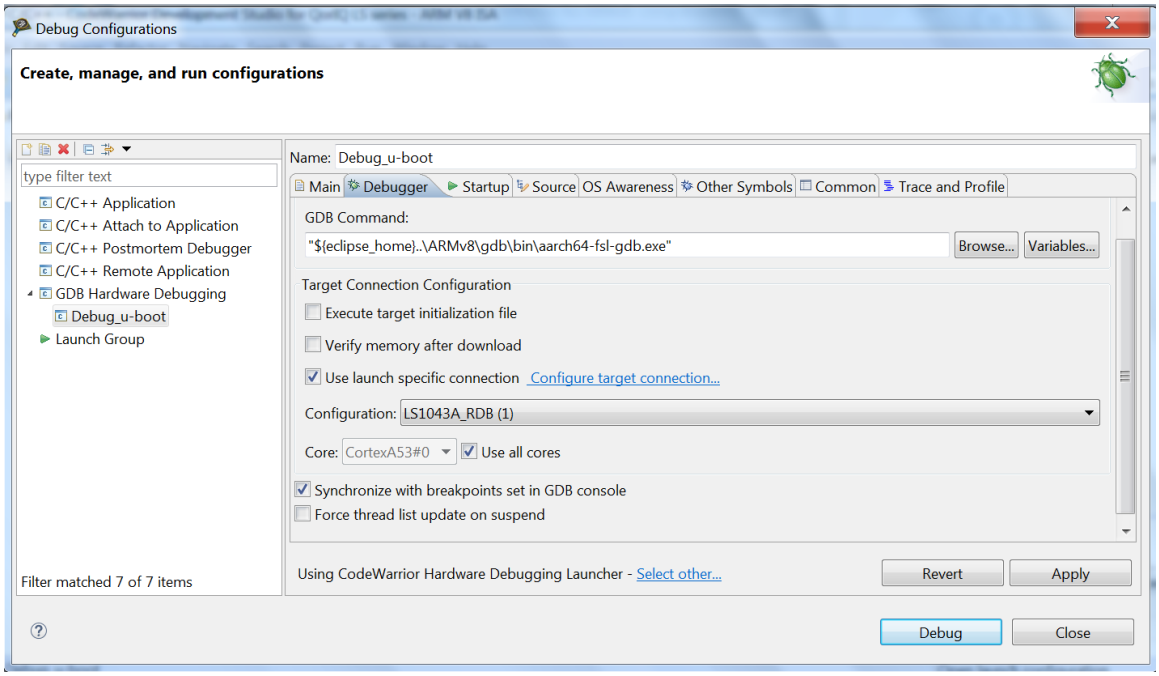
$bitbake u-boot -c cleansstate
$bitbake u-boot

This document discuss debugging u-boot with attach method, so please program u-boot-nor-2016.01+fslgit-r0.bin to NOR flash first as mentioned in section 1.

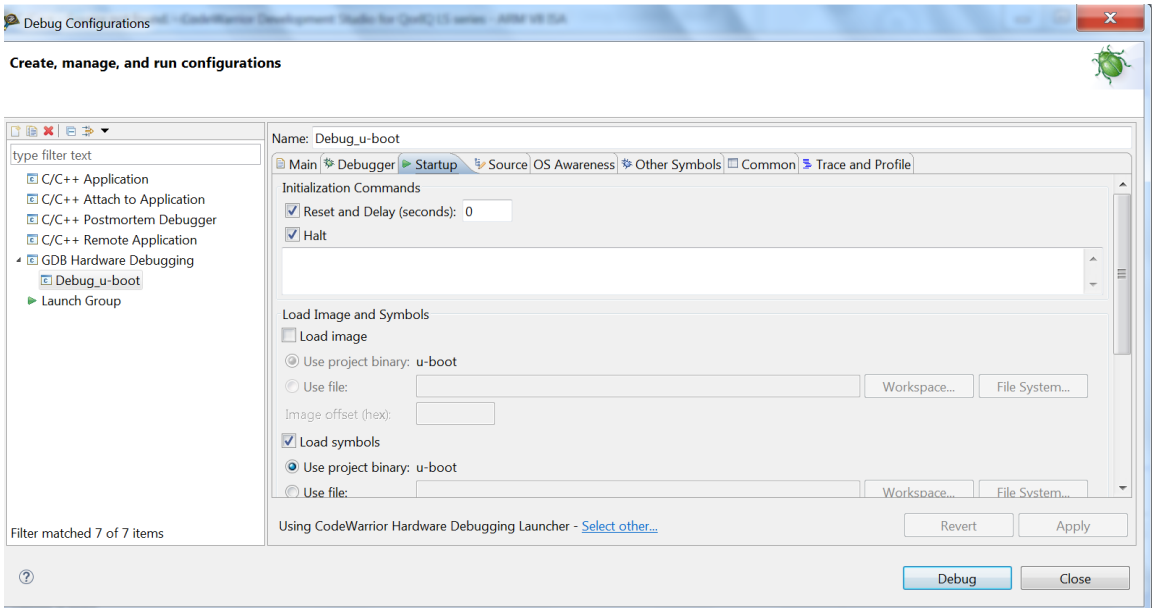
Import “u-boot” in ls1043ardb_config folder to CodeWarrior IDE from File->New->CodeWarrior Executable Importer.



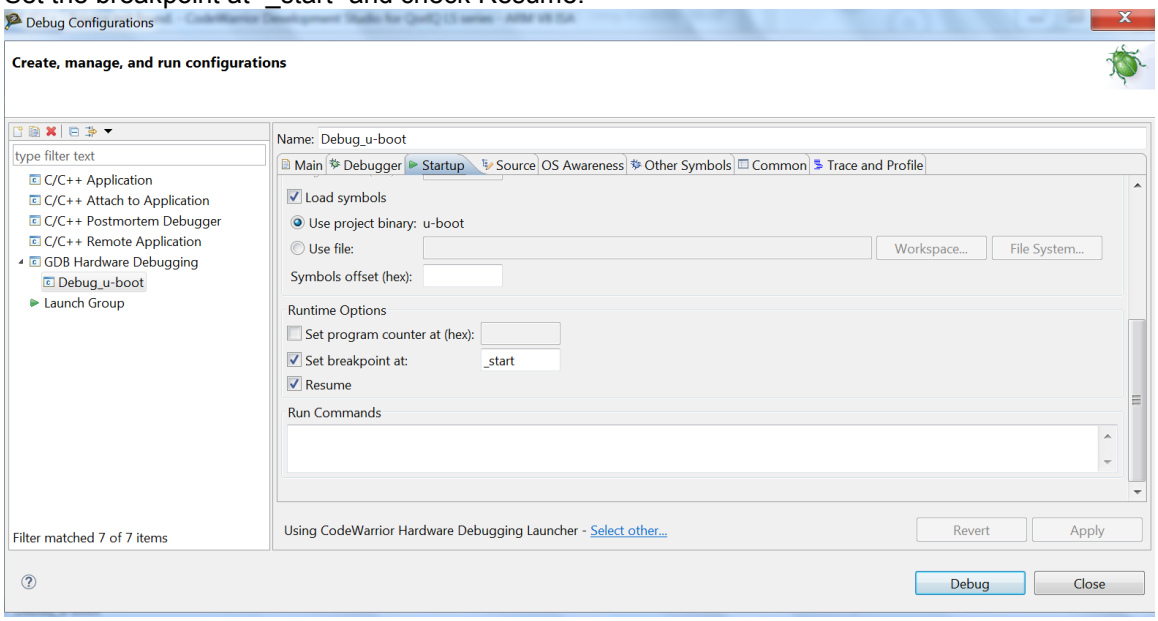
Debug Configuration is configured as the following, target launch connection should be configured as mentioned in section 1.



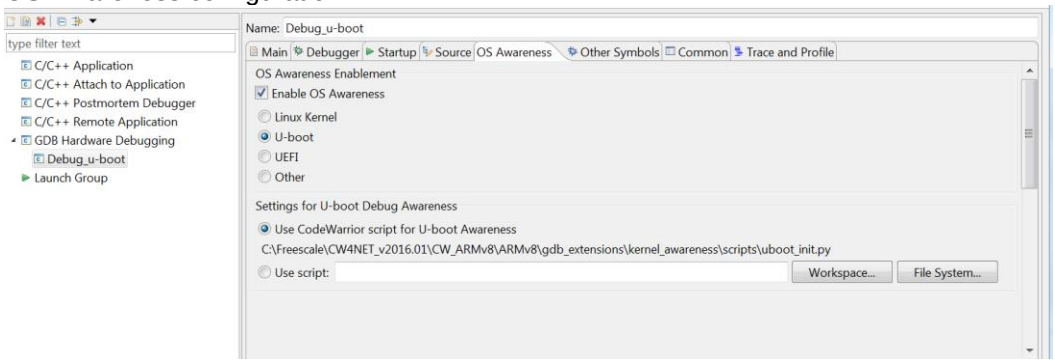
In the Startup panel, configure “Reset and Delay” as 0, uncheck “Load image” to use CodeWarrior to attach to the running software on the target board.



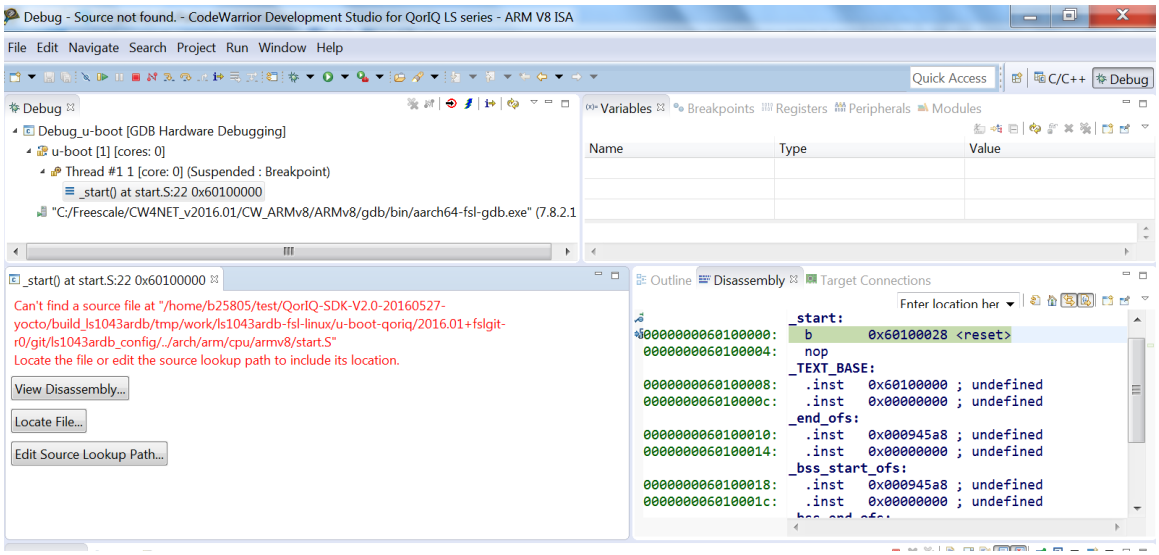
Set the breakpoint at “_start” and check Resume.



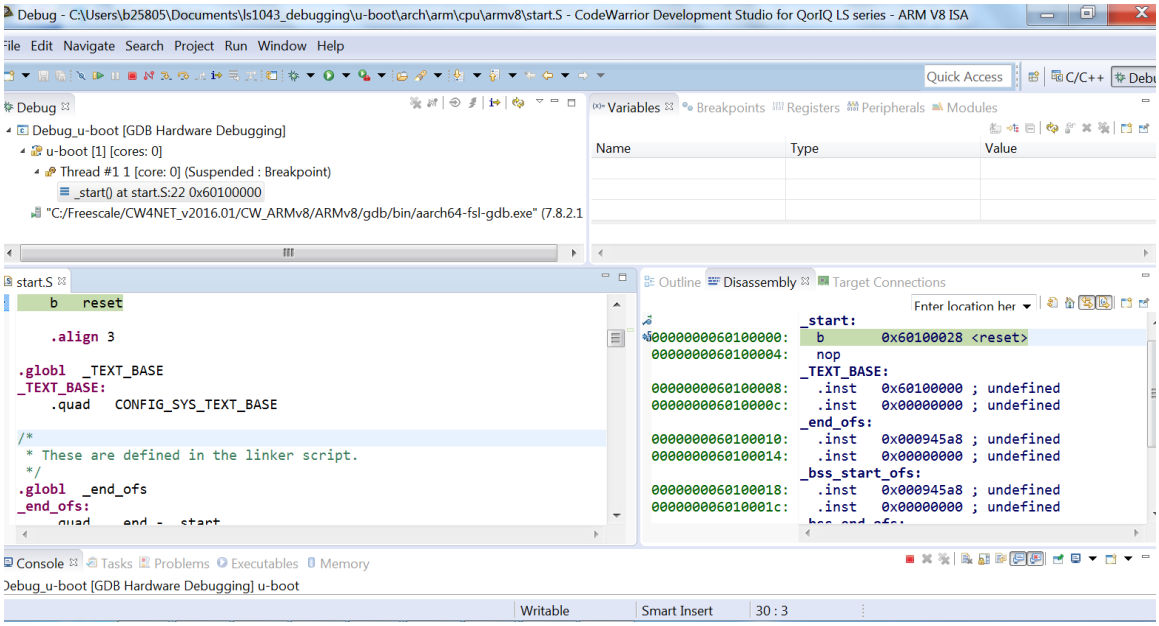
OS Awareness configuration



Click Debug to connect to the target, the debugger stops at _start symbol, double click "_start() at start.S" in the Debug Window to view the source code.



Click "Locate File" to relocate the source file



3. Debug Linux Kernel by Attaching Running U-boot

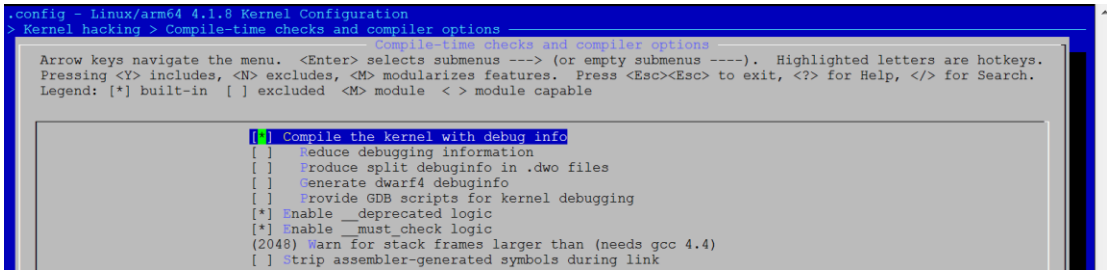
Build Linux Kernel with CodeWarrior debug options enabled in SDK 2.0 Yocto environment. Fetch Linux Kernel source and rebuild image as the following.

$bitbake virtual/kernel -c cleansstate

$bitbake virtual/kernel -c configure

Go to Kernel source folder build_ls1043ardb/tmp/work/ls1043ardb-fsl-linux/linux-qoriq/4.1-r0/build, configure Kernel with debug option enabled.

$ make ARCH=arm64 menuconfig



Rebuild Linux Kernel

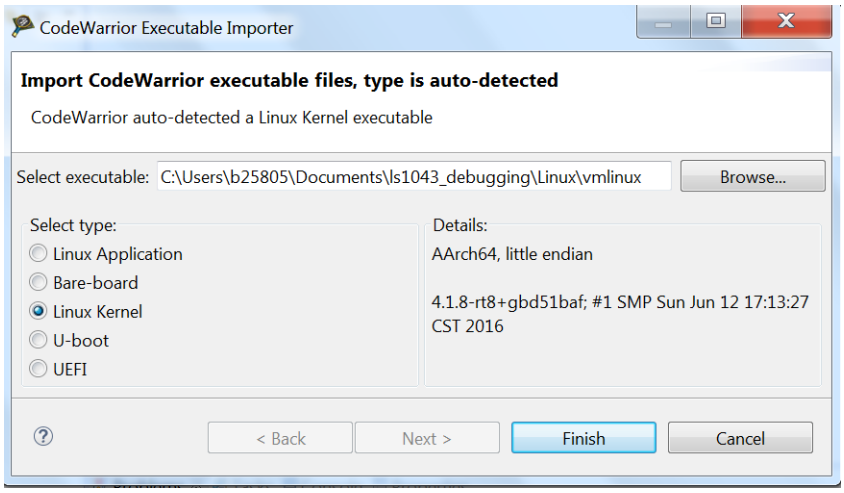
$ bitbake virtual/kernel

$ bitbake fsl-image-kernelitb

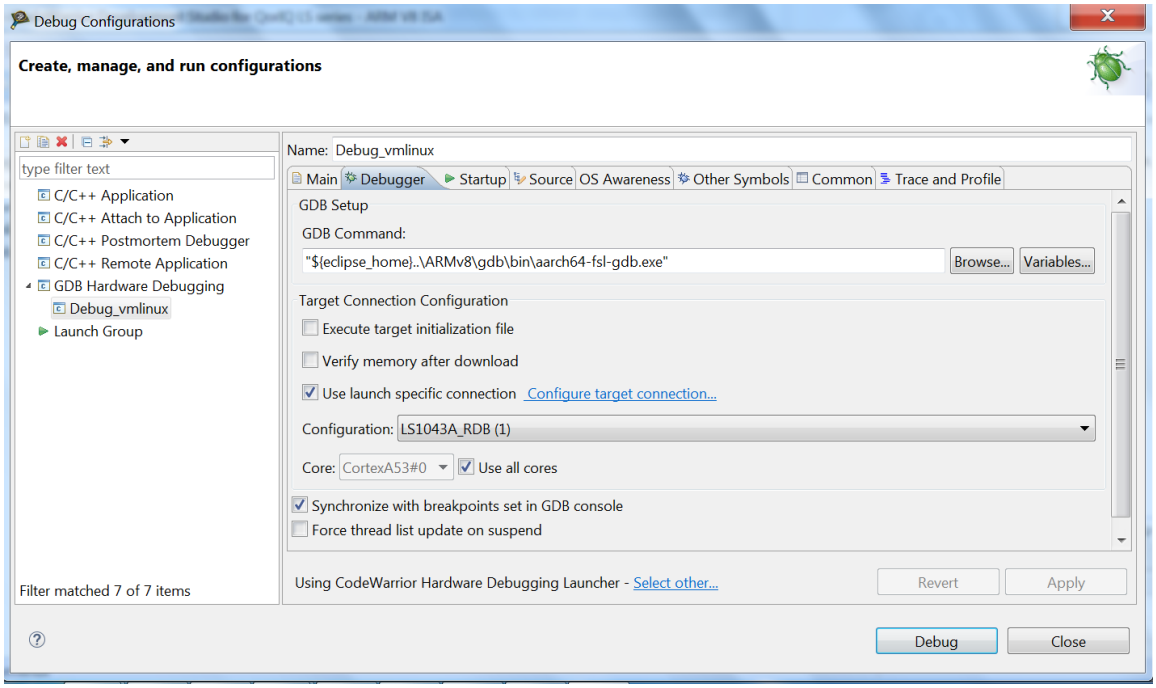
Copy the Linux Kernel source folder build_ls1043ardb/tmp/work-shared/ls1043ardb/kernel-source to the debug machine, prepare build_ls1043ardb/tmp/deploy/images/ls1043ardb/kernel-fsl-ls1043a-rdb.itb to download to the target board for debugging.

The following procedure introduces how to configure CodeWarror to debug Linux Kernel by attaching u-boot.

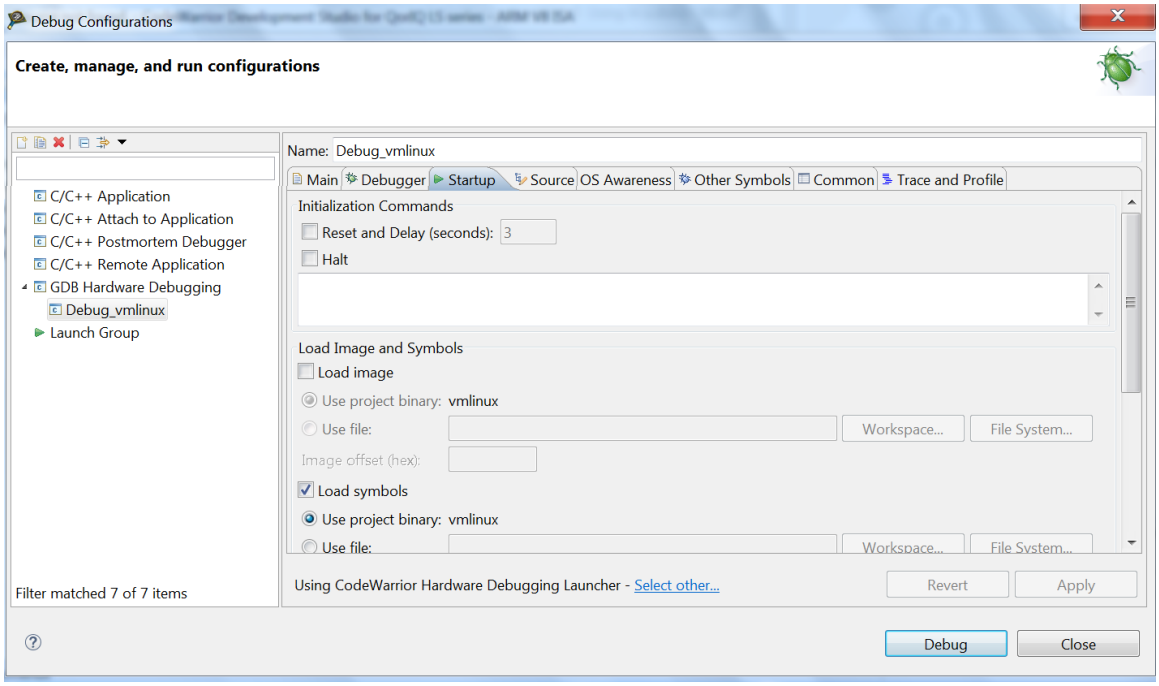
Import vmlinux to CodeWarrior IDE from File->New->CodeWarrior Executable Importer.

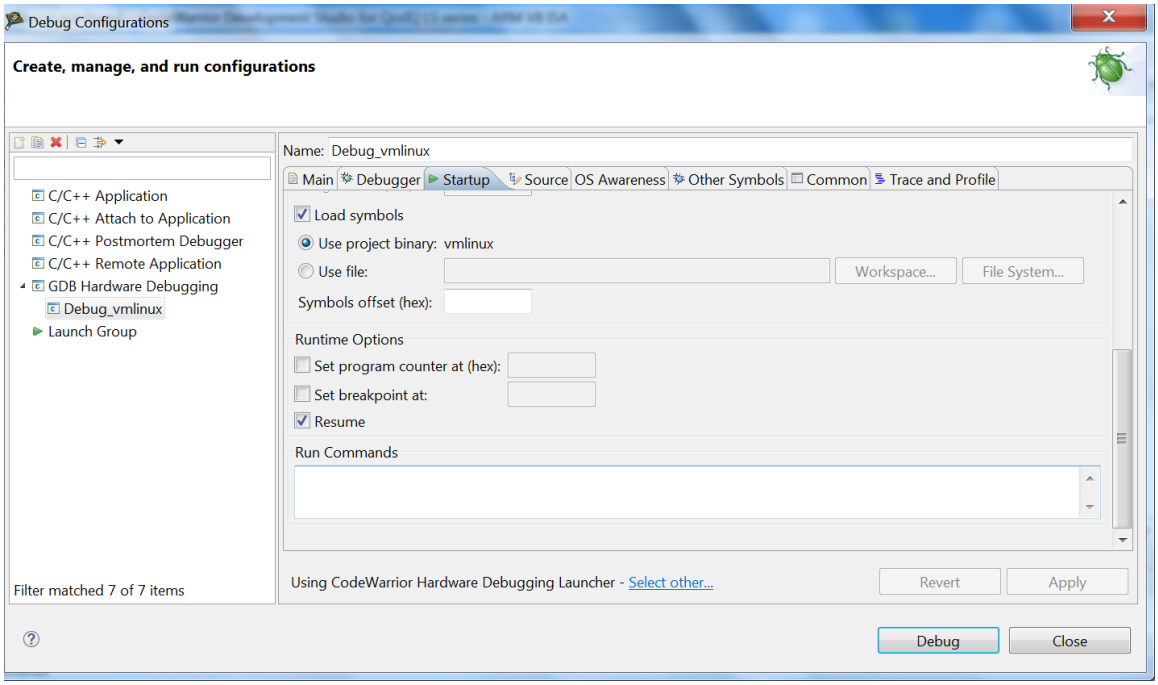


Configure Debugger

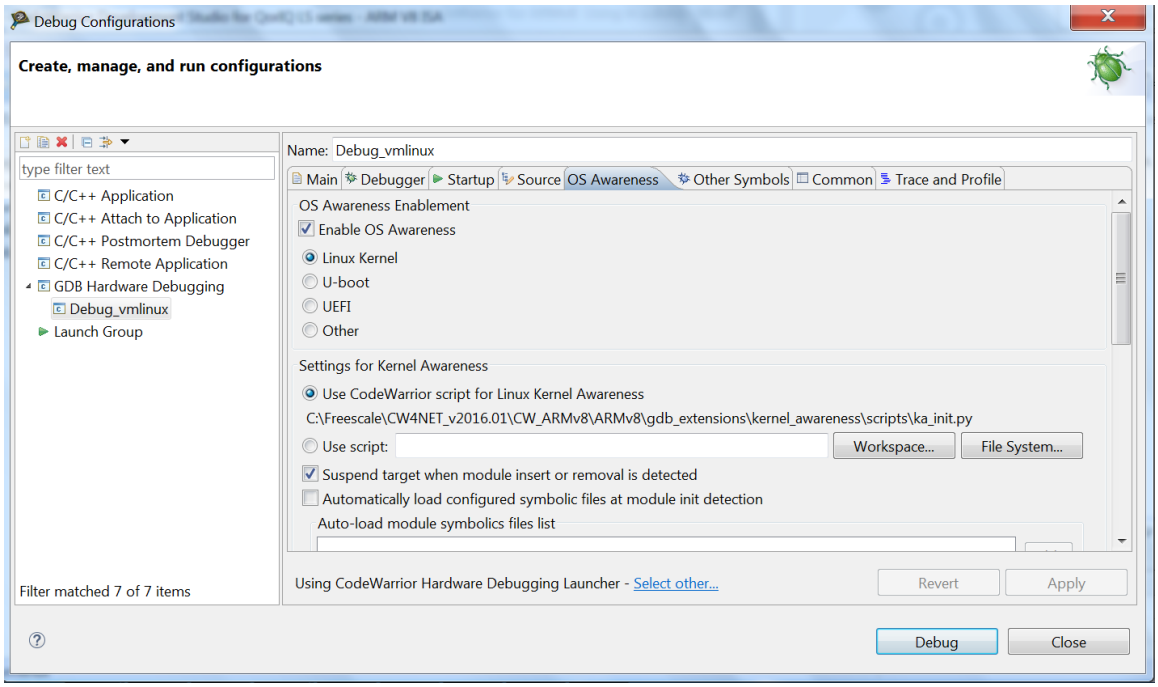


Configure Startup panel

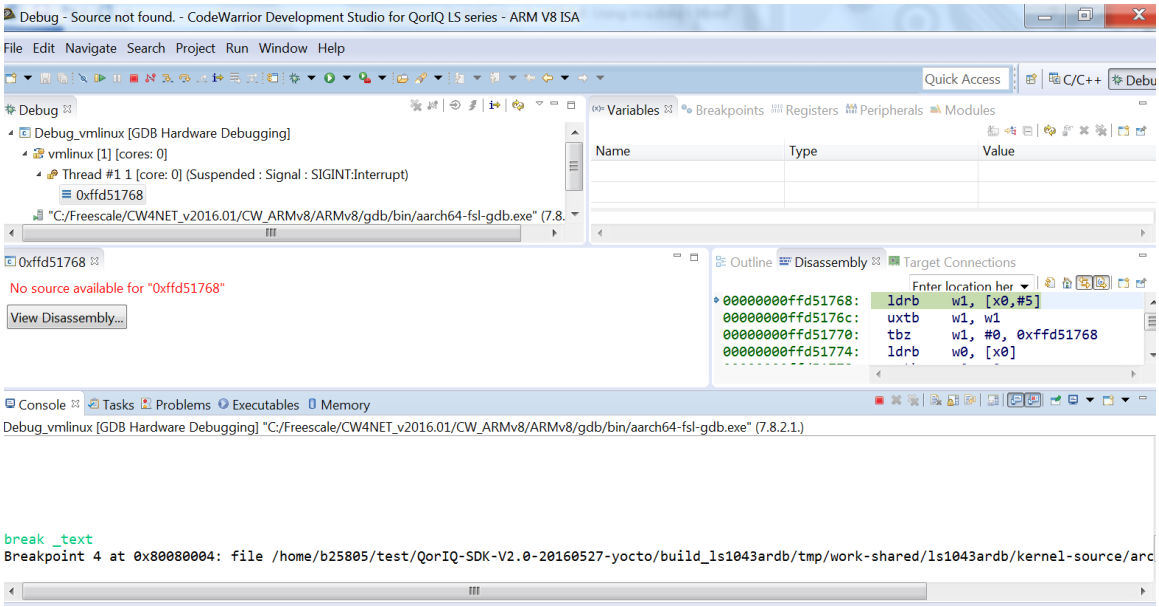




Configure OS Awareness panel



On the target board, after setting up u-boot and entering into u-boot prompt. In CodeWarrior IDE, click “Debug” to connect to the target board to attach to u-boot. After CodeWarrior connect to the target board, please click “Suspend” button in CodeWarrior IDE to stop the running u-boot, and enter “break _text” in the gdb console to set the breakpoint at the start of Linux Kernel.

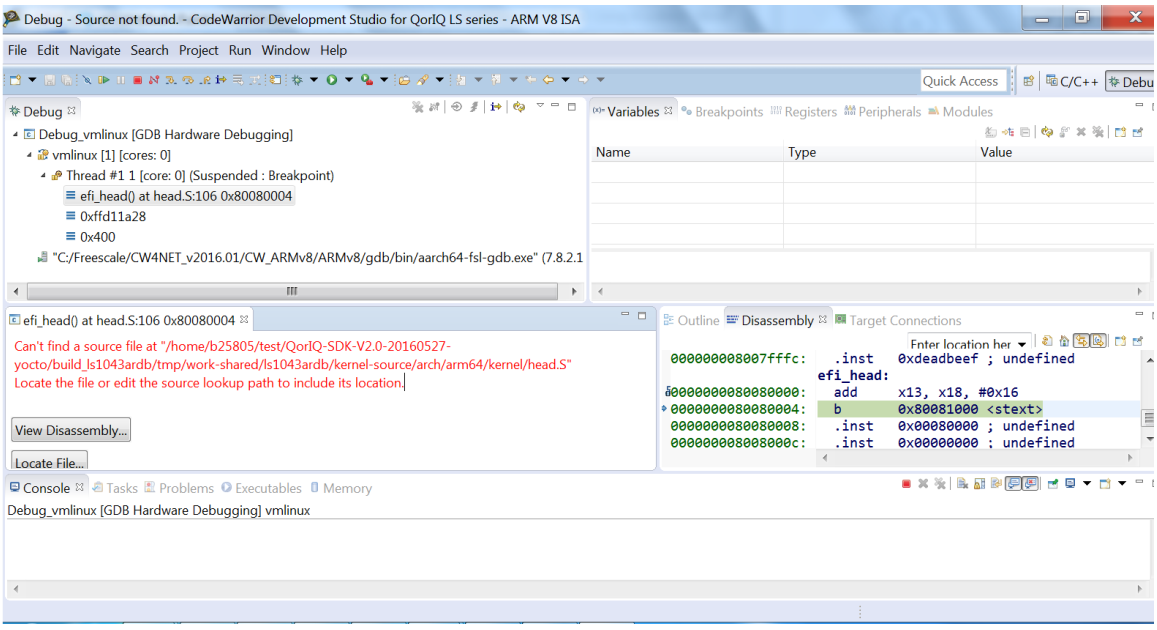


In CodeWarrior IDE resume the program, in the target console, boot up Linux Kernel.

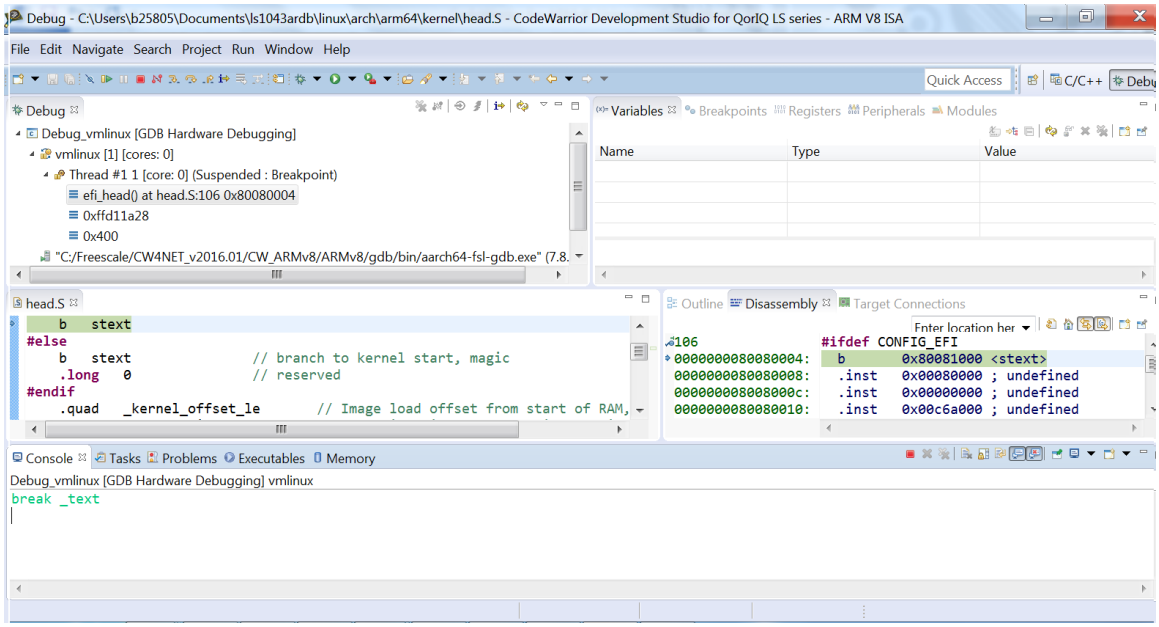
=> tftp 0xa0000000 b25805/ls1043rdb/kernel-fsl-ls1043a-rdb.itb

=> bootm 0xa0000000

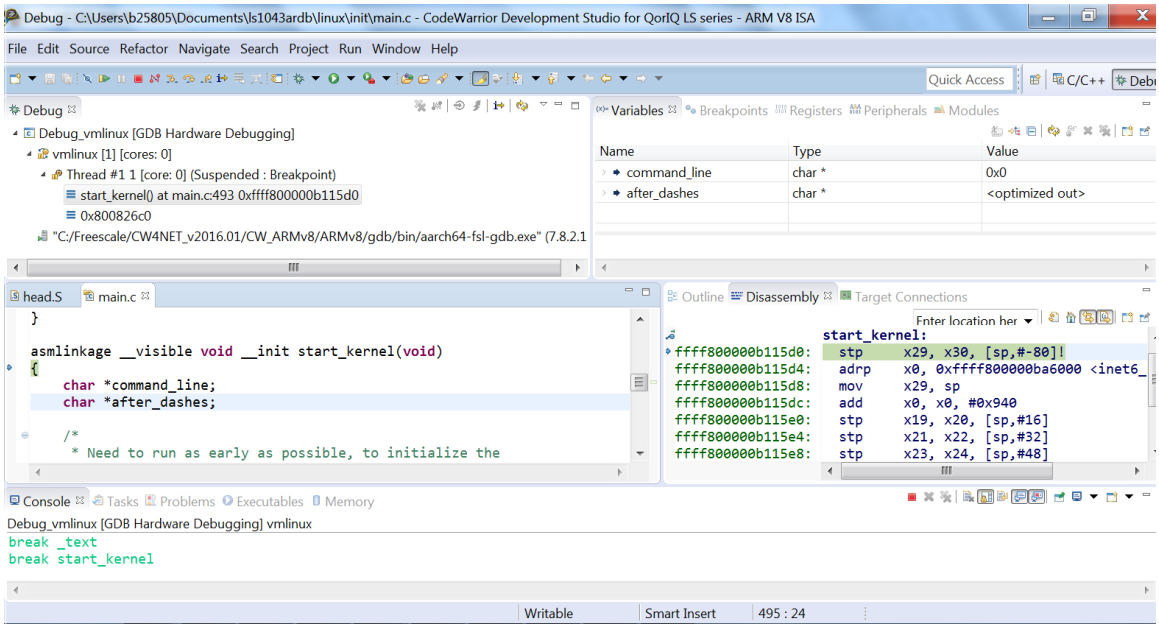
In CodeWarrior IDE, the debugger will stop at the Linux Kernel entry point address.



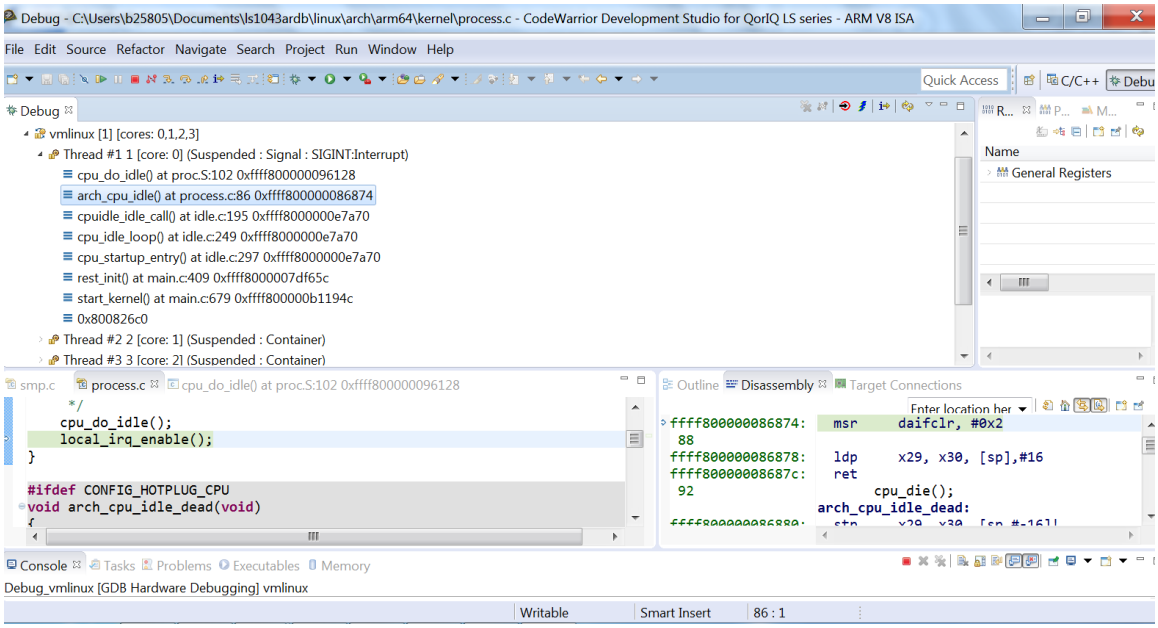
Locate the source file and debug Linux Kernel at the entry point.



In the GDB console enter “break start_kernel” and click Resume button to continue to debug Linux Kernel after MMU is enabled.



CodeWarrior lists the status for all cores in SMP software after all secondary cores been set up.


Debug - C:\Users\b25805\Documents\ls1043ardb\linux\arch\arm64\kernel\process.c - CodeWarrior Development Studio for QorIQ LS series - ARM V8 ISA
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
C/C++
Debug
vmlinux [1] [cores: 0,1,2,3]
Thread #1 1 [core: 0] (Suspended : Signal : SIGINT:Interrupt)
cpu_do_idle() at proc.S:102 0xffff800000096128
arch_cpu_idle() at process.c:86 0xffff800000086874
cpuidle_idle_call() at idle.c:195 0xffff8000000e7a70
cpu_idle_loop() at idle.c:249 0xffff8000000e7a70
cpu_startup_entry() at idle.c:297 0xffff8000000e7a70
rest_init() at main.c:409 0xffff8000007df65c
start_kernel() at main.c:679 0xffff800000b1194c
0x800826c0
Thread #2 2 [core: 1] (Suspended : Container)
Thread #3 3 [core: 2] (Suspended : Container)
Name
General Registers
smp.c
process.c
cpu_do_idle() at proc.S:102 0xffff800000096128
*/
cpu_do_idle();
local_irq_enable();
}
#ifdef CONFIG_HOTPLUG_CPU
void arch_cpu_idle_dead(void)
Outline
Disassembly
Target Connections
ffff800000086874: msr daifclr, #0x2
88
ffff800000086878: ldp x29, x30, [sp],#16
ffff80000008687c: ret
92 cpu_die();
arch_cpu_idle_dead:
Console
Tasks
Problems
Executables
Memory
Debug_vmlinux [GDB Hardware Debugging] vmlinux
Writable
Smart Insert
86 : 1