

Getting Started with Freescale MQX™ RTOS for Kinetis SDK and ARM® GCC

1 Overview

This section describes the steps required to configure ARM® GCC and Cmake to build, run, and debug MQX™ RTOS demo applications and necessary driver libraries provided in the KSDK framework. The Hello World demo application targeted for the TWR-K64F120M Tower System hardware platform is used as an example in this guide.

Contents

1	Overview	1
2	Environment Setup	2
2.1	Windows® operating system	2
2.2	Linux® OS	4
3	Run a demo application	5
3.1	Build a demo application and required libraries	5
3.2	Run a demo application	6
4	Revision history	14

2 Environment Setup

2.1 Windows® operating system

2.1.1 Install MinGW

1. Download and install the installer from www.launchpad.net/gcc-arm-embedded. See the MQX RTOS Release Notes document for supported ARM GCC versions.
2. Install the GCC ARM Embedded tool chain from <http://www.mingw.org>. The recommended path is C:/MINGW, but you can install to any location. Note that the installation path may not contain a space.
3. Ensure that the mingw32-base and msys-base are selected under basic setup.
4. Click “Installation” and select “Apply changes”.
5. Add paths C:/MINGW/msys/1.0/bin and C:/MINGW/bin to the system environment. Note that if the GCC ARM Embedded tool chain is installed somewhere else other than the recommended location, the system paths added will reflect this change. An example using the recommended installation location is shown.

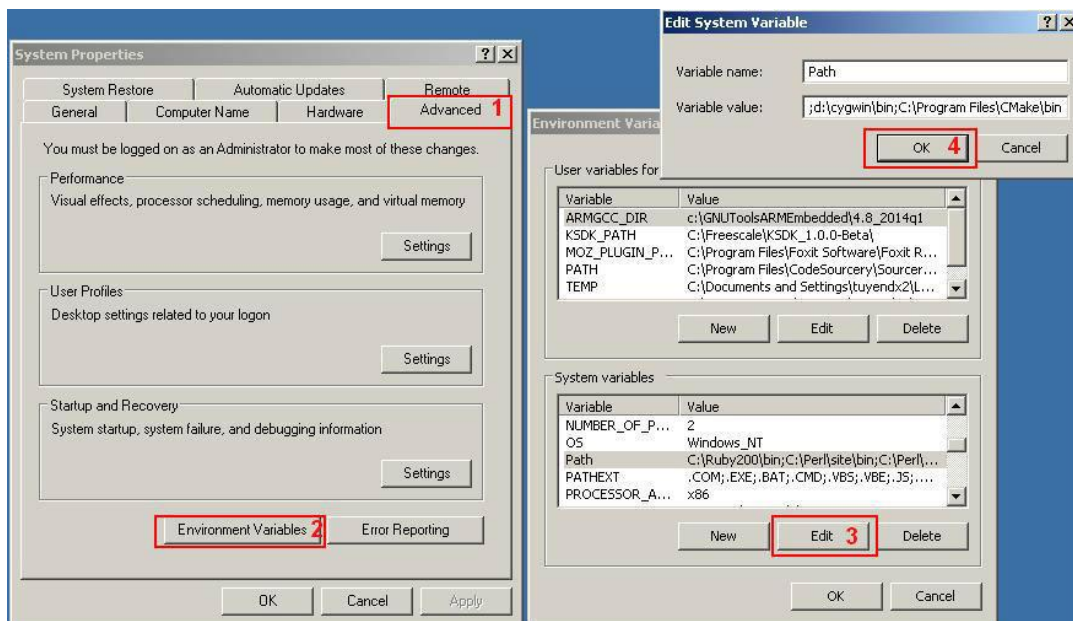


Figure 1: Add Path to systems environment example for Windows® XP

2.1.2 Add new system environment variable ARMGCC_DIR

1. Go to the “Properties” tab on your computer.

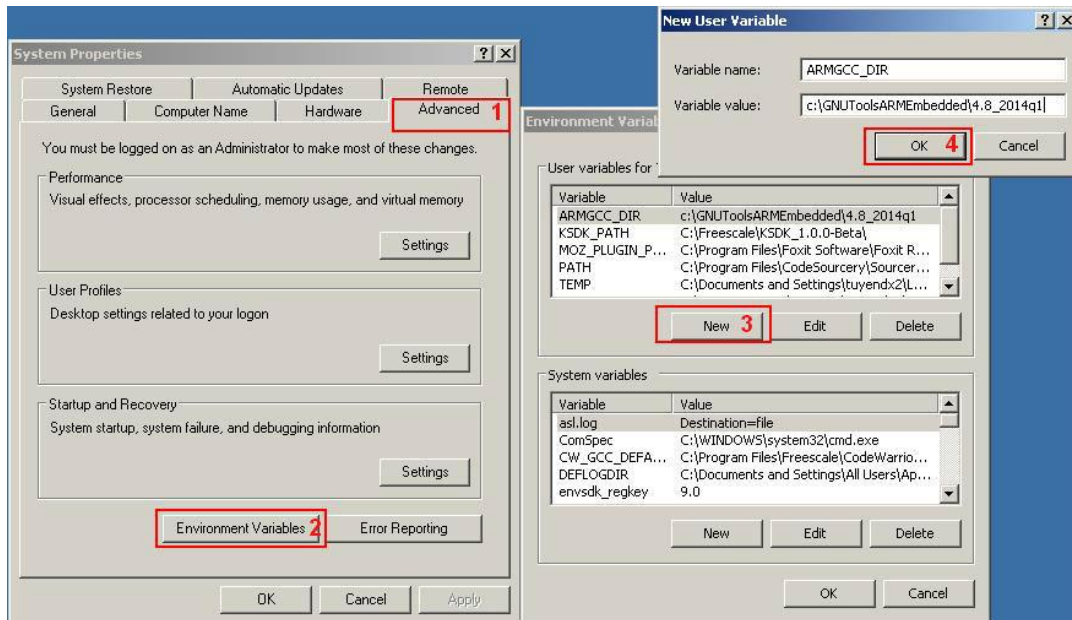


Figure 2: Add variable ARMGCC_DIR example for Windows XP

The variable name as ARMGCC_DIR, and the value as a path to the installation directory of the ARM GCC tool chain (ensure that a path does not contain spaces).

2.1.3 Install CMake

1. Download and install CMake from www.cmake.org/cmake/resources/software.html. See the MQX RTOS Release Notes for supported CMake versions. Ensure the option "Add CMake to system PATH" is selected when installing.



Figure 3: install CMake

2.2 Linux® OS

2.2.1 Setup ARMGCC tool on Linux OS

All tool files necessary setup for ARMGCC:

cmake-3.1.0-rc1-Linux-i386.tar.gz

gcc-arm-none-eabi-4_8-2014q3-20140805-linux.tar.bz2

2.2.2 Add environment variables

1. Go to the `/home/public/` folder. Press Ctrl + H.
2. Open the `.bashrc` file.
3. Go to the end of this file and add these lines:

```
export PATH="/home/public/Tools/cmake-3.1.0-rc1-Linux-i386/bin":$PATH
export ARMGCC_DIR="/home/public/Tools/gcc-arm-none-eabi-4_8-2014q3"
```

3 Run a demo application

3.1 Build a demo application and required libraries

Follow the instructions to build the demo application and required libraries. The batch file distributed with the MQX RTOS release automatically builds ksdk_mqx_lib, mqx, and mqx_stdlib before the example application.

1. Open a Windows operating system command interpreter (cmd.exe). Open bash in Linux OS.
2. Change the directory to the example application directory.

```
<install_dir>/rtos/mqx/mqx/examples/<demo_name>/build/armgcc/<demo_name>_<board_name>
```

3. Execute the build batch files.
 - Windows operating system: Type build_int_flash_debug or build_int_flash_release for build with the Debug or Release target.
 - Linux OS: Executes mass build scripts./build_int_flash_debug or ./build_int_flash_release to build appropriate build targets.

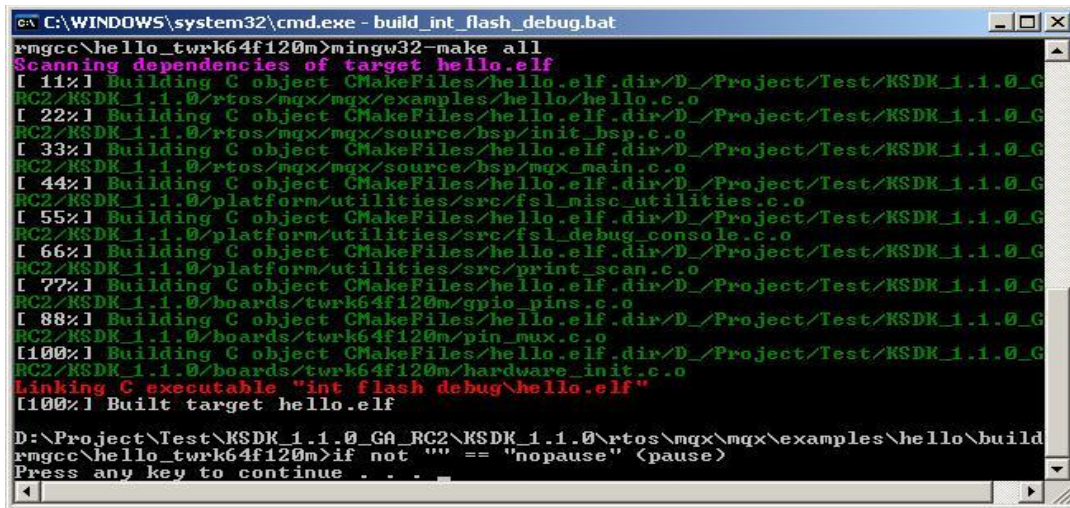
These archive files are generated after a successful build of the library project files.

```
<install_dir>/lib/ksdk_mqx_lib/armgcc/<device_name>/<build_target>/<libksdk_platform_mqx>.a
```

```
<install_dir>/rtos/mqx/lib/<board_name>.armgcc/<build_target>/mqx/<lib_mqx>.a
```

```
<install_dir>/rtos/mqx/lib/<board_name>.armgcc/<build_target>/mqx_stdlib/<lib_mqx_stdlib>.a
```

This output message is shown when armgcc finishes building the hello demo application for K64F120M.



```
C:\WINDOWS\system32\cmd.exe - build_int_flash_debug.bat
rmgcc\hello_twrk64f120m>mingw32-make all
Scanning dependencies of target hello.elf
[ 11%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/rtos/mqx/mqx/examples/hello/hello.c.o
[ 22%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/rtos/mqx/mqx/source/bsp/init_bsp.c.o
[ 33%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/rtos/mqx/mqx/source/bsp/mqx_main.c.o
[ 44%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/platform/utilities/src/fsl_misc_utilities.c.o
[ 55%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/platform/utilities/src/fsl_debug_console.c.o
[ 66%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/platform/utilities/src/print_scan.c.o
[ 77%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/boards/twrk64f120m/gpio_pins.c.o
[ 88%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/boards/twrk64f120m/pin_mux.c.o
[100%] Building C object CMakeFiles/hello.elf.dir/D:/Project/Test/KSDK_1.1.0_G
RC2/KSDK_1.1.0/boards/twrk64f120m/hardware_init.c.o
Linking C executable "int flash debug\hello.elf"
[100%] Built target hello.elf

D:\Project\Test\KSDK_1.1.0_GA_RC2\KSDK_1.1.0\rtos\mqx\mqx\examples\hello\build
rmgcc\hello_twrk64f120m>if not "" == "nopause" (pause)
Press any key to continue . . .
```

Figure 4: Build ksdk-mqx demo output message

3.2 Run a demo application

This section describes the steps to run a demo application using the J-Link GDB Server application.

To download and run the application, perform these steps:

1. Connect the development platform to your PC via USB cable between the OpenSDA USB connector and the PC USB connector.

2. Open the terminal application, such as PuTTY or TeraTerm, on the PC, and connect to the OpenSDA serial port number. Configure the terminal with these settings:
 - a) 115200 baud rate
 - b) No parity
 - c) 8 data bits
 - d) 1 stop bit

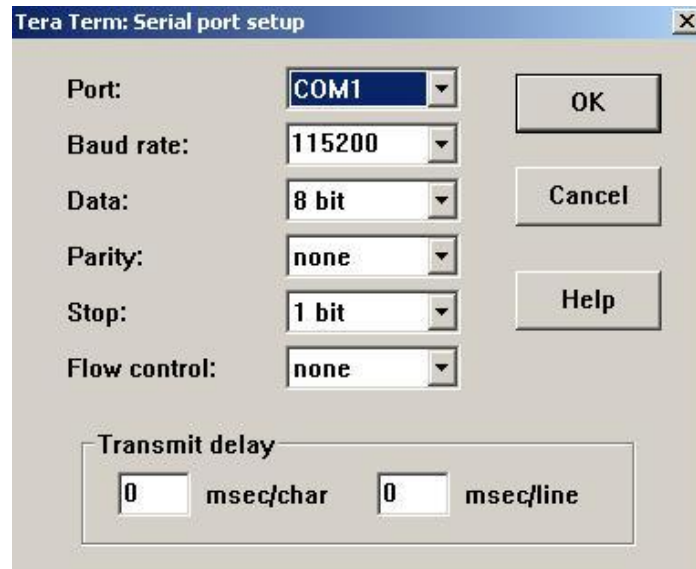


Figure 5 Terminal configuration on Windows operating system

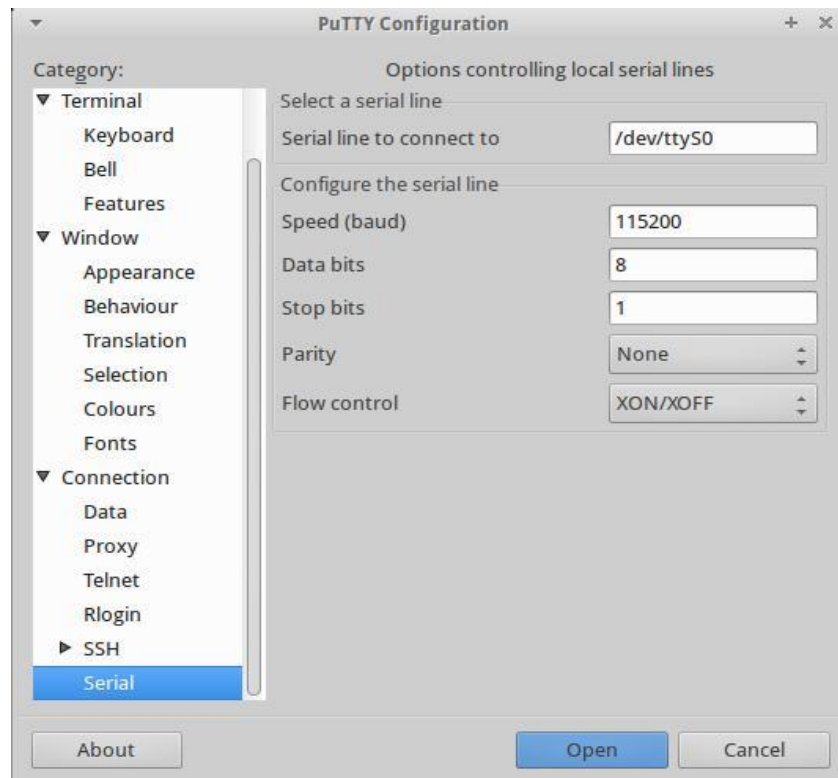


Figure 6: Terminal configuration on Linux OS

3. Open the J-Link GDB Server application and setting:

- Windows operating system

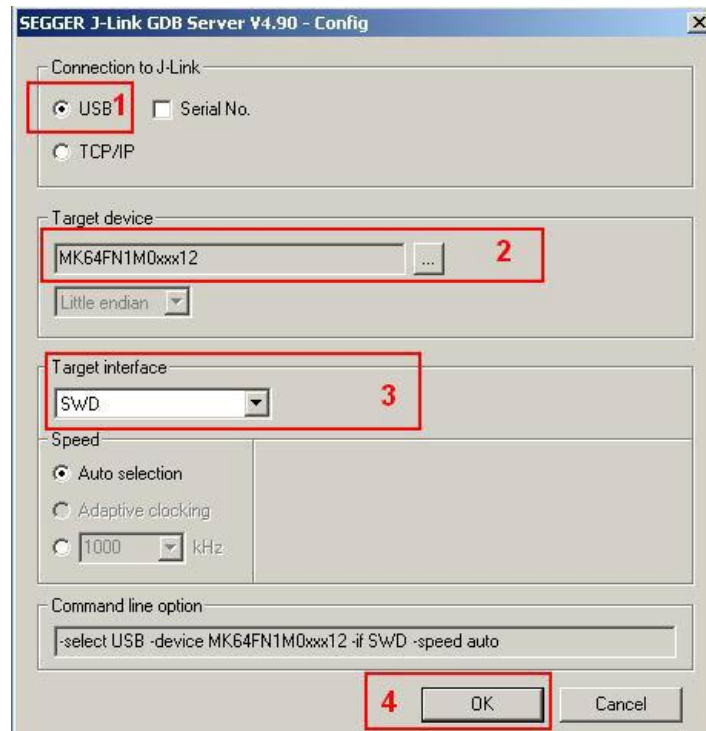


Figure 7: Setting J-Link GDB Server on Windows operating system

- Linux OS

4. Open bash and execute this command:

JLinkGDBServer -device <ChipName> -if <SWD/JTAG> -speed 1000 -endian little.

4. Open bash and execute this command:

```
JLinkGDBServer -device <ChipName> -if <SWD/JTAG> -speed 1000 -endian little
```

Once connected, the screen should resemble this figure.

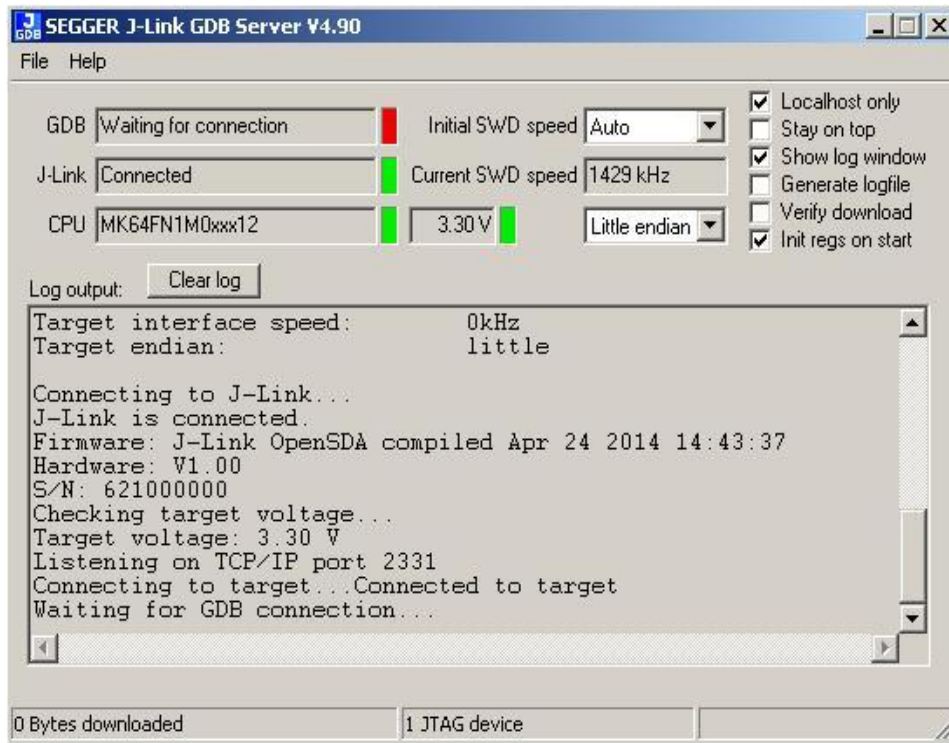
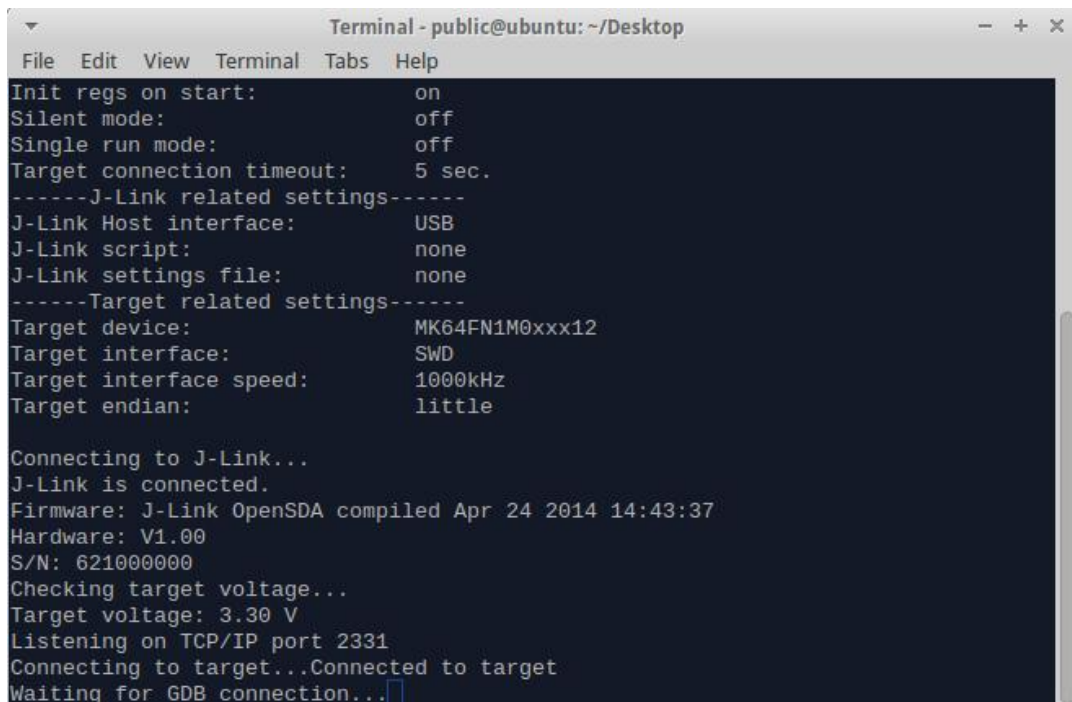


Figure 8: J-Link GDB Server on Windows operating system

A terminal window titled "Terminal - public@ubuntu: ~/Desktop" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal output shows the J-Link GDB Server configuration and connection status. It lists settings for J-Link (Host interface: USB, script: none, settings file: none) and Target (device: MK64FN1M0xxx12, interface: SWD, speed: 1000kHz, endian: little). It then shows the connection process: "Connecting to J-Link...", "J-Link is connected.", "Firmware: J-Link OpenSDA compiled Apr 24 2014 14:43:37", "Hardware: V1.00", "S/N: 621000000", "Checking target voltage...", "Target voltage: 3.30 V", "Listening on TCP/IP port 2331", "Connecting to target...Connected to target", and "Waiting for GDB connection...".

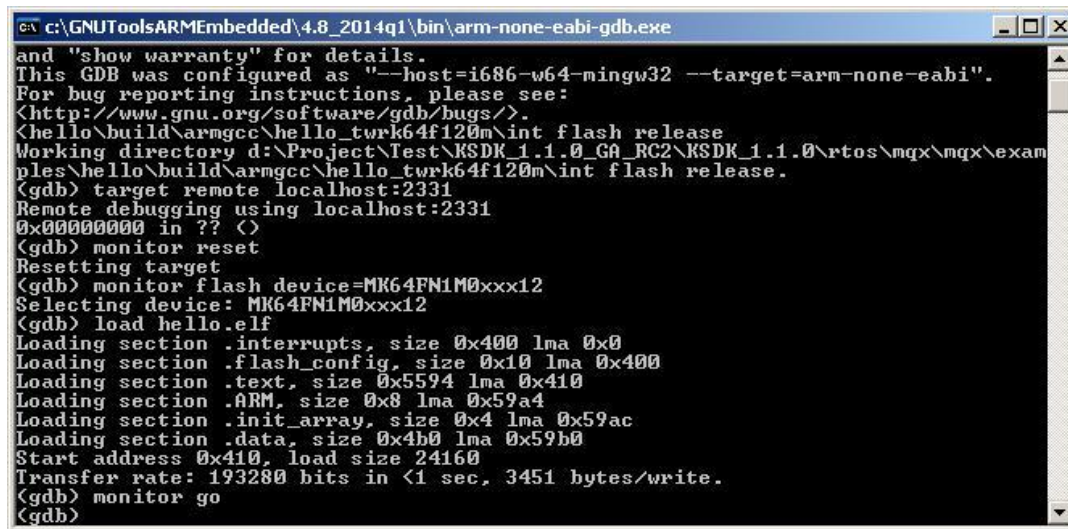
```
Init regs on start:      on
Silent mode:            off
Single run mode:        off
Target connection timeout: 5 sec.
-----J-Link related settings-----
J-Link Host interface:  USB
J-Link script:          none
J-Link settings file:   none
-----Target related settings-----
Target device:          MK64FN1M0xxx12
Target interface:       SWD
Target interface speed: 1000kHz
Target endian:          little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link OpenSDA compiled Apr 24 2014 14:43:37
Hardware: V1.00
S/N: 621000000
Checking target voltage...
Target voltage: 3.30 V
Listening on TCP/IP port 2331
Connecting to target...Connected to target
Waiting for GDB connection...
```

Figure 9: J-Link GDB Server on Linux OS

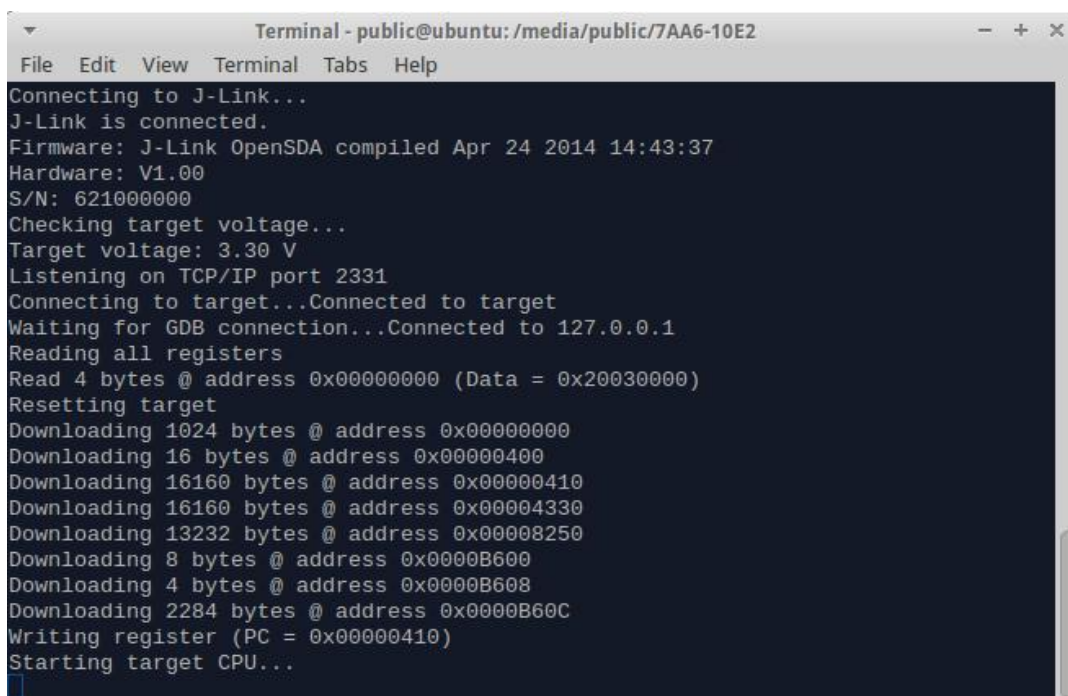
5. Open a Windows operating system command interpreter (cmd.exe) in Linux OS open bash and change the directory to the output directory of the desired demo. For example:
`<install_dir>/rtos/mqx/mqx/example/hello/build/armgcc/hello_<board_name>/<build_target>`
6. Run these commands:
 - a) `arm-none-eabi-gdb.exe`
 - b) `target remote localhost:2331`
 - c) `monitor reset`
 - d) `monitor flash device=<chip_name>` (Example: `monitor flash device=MK64FN1M0xxx12`)
 - e) `load <demo_name>.elf` (Example: `load hello.elf`)

7. Once the application downloads, execute “Monitor go” to start demo application.



```
c:\GNUToolsARMEEmbedded\4.8_2014q1\bin\arm-none-eabi-gdb.exe
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
<hello\build\armgcc\hello_twrk64f120m\int flash release
Working directory d:\Project\Test\KSDK_1.1.0_GA_RC2\KSDK_1.1.0\rtos\mqx\mqx\exam
ples\hello\build\armgcc\hello_twrk64f120m\int flash release.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00000000 in ?? (<)
(gdb) monitor reset
Resetting target
(gdb) monitor flash device=MK64FN1M0xxx12
Selecting device: MK64FN1M0xxx12
(gdb) load hello.elf
Loading section .interrupts, size 0x400 lma 0x0
Loading section .flash_config, size 0x10 lma 0x400
Loading section .text, size 0x5594 lma 0x410
Loading section .ARM, size 0x8 lma 0x59a4
Loading section .init_array, size 0x4 lma 0x59ac
Loading section .data, size 0x4b0 lma 0x59b0
Start address 0x410, load size 24160
Transfer rate: 193280 bits in <1 sec, 3451 bytes/write.
(gdb) monitor go
(gdb)
```

Figure 10: Run command to load demo application on Windows operating system



```
Terminal - public@ubuntu: /media/public/7AA6-10E2
File Edit View Terminal Tabs Help
Connecting to J-Link...
J-Link is connected.
Firmware: J-Link OpenSDA compiled Apr 24 2014 14:43:37
Hardware: V1.00
S/N: 621000000
Checking target voltage...
Target voltage: 3.30 V
Listening on TCP/IP port 2331
Connecting to target...Connected to target
Waiting for GDB connection...Connected to 127.0.0.1
Reading all registers
Read 4 bytes @ address 0x00000000 (Data = 0x20030000)
Resetting target
Downloading 1024 bytes @ address 0x00000000
Downloading 16 bytes @ address 0x00000400
Downloading 16160 bytes @ address 0x00000410
Downloading 16160 bytes @ address 0x00004330
Downloading 13232 bytes @ address 0x00008250
Downloading 8 bytes @ address 0x0000B600
Downloading 4 bytes @ address 0x0000B608
Downloading 2284 bytes @ address 0x0000B60C
Writing register (PC = 0x00000410)
Starting target CPU...
```

Figure 11: Run command to load demo application on Linux OS



Figure 12: Output of hello_world

4 Revision History

This table summarizes revisions to this document.

Table 1 Revision History		
Revision number	Date	Substantial changes
1	04/2015	Kinetis SDK 1.2.0 release
0	12/2014	Kinetis SDK 1.1.0 release

How to Reach Us:**Home Page:**

freescale.com

Web Support:

freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, and Kinetis are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners. ARM, ARM powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2015 Freescale Semiconductor, Inc.

