Document Number: MQXGDBUG Rev. 1, 08/2014

Using Eclipse and GDB with Freescale MQX[™] RTOS User's Guide



How to Reach Us:

Home Page: www.freescale.com

Web Support: http://www.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. Vybrid and Tower are trademarks of Freescale Semiconductor, Inc. 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.

© 2008-2014 Freescale Semiconductor, Inc.



Table of Contents

Using Eclipse and GDB with Freescale MQX™ RTOS User's Guide	i
1 Read Me First	2
2 MQX Build – Initial Steps	3
2.1 Building the MQX software using a command line 2.2 Makefile build	
3 Using MQX Makefiles with Eclipse	6
4 Running and Debugging the MQX Application Using J-Link Gdbserver	10
4.1 Starting the gdbserver - Windows4.2 Starting the gdbserver - Linux4.3 Using Eclipse with the gdbserver	12
5 Makefiles Structure	18

1 Read Me First

This document describes the steps to configure the GNU and Eclipse CDT development tools version 4.3 Kepler and use them to build, run, and debug applications of the Freescale MQXTM RTOS operating system. See *Getting Started with Freescale MQXTM RTOS* (document MQXGSRTOS) and other user documentation included with the latest Freescale MQX RTOS installation for more details not specifically related to the GNU and Eclipse CDC tools.

Get the latest Freescale MQX RTOS at freescale.com/mqx.

2 MQX Build – Initial Steps

The MQX software release provides the makefiles to build MQX software libraries and applications either from the Windows[®] operating system or the Linux command line. Makefiles can also be integrated with the integrated development environments (IDEs). This document describes integration with the Eclipse IDE. These are the settings to prepare your build environment:

Windows operating system – a common scenario is to use "mingw" utilities with the Windows "cmd" utility.

- Install mingw from sourceforge.net/projects/mingw/ to a default location "c:\MinGW".
- Ensure that your PATH variable contains "c:\MinGW\bin".
- Open the MQX installation directory, which is located in the C:/Freescale/Freescale MQX 4.1 by default. Edit the "build/common/make/global.mak" to set up a valid compiler directory path to a variable TOOLCHAIN_ROOTDIR. The file contains some commented examples for each toolchain. Because the path cannot contain whitespaces, use a Windows command utility to get a DOS path without spaces. The list of supported tool-chains is located in the Release Notes.

Linux operating system:

- Install "make" and "sed" utilities. Use your Linux distribution package manager to get the latest version of the required tools.
- Open the MQX installation directory and edit the file "build/common/make/global.mak" to set up a valid cross compiler directory path to a variable TOOLCHAIN_ROOTDIR. The file contains some commented examples for each toolchain. Note that the path cannot contain whitespaces.

Note that the instructions in the subsequent sections apply to the TWR-K60N512 BSP and the Hello World example application. The same instructions apply for all other BSPs and examples.

2.1 Building the MQX software using a command line

See Chapter 2 of the *Getting Started with Freescale MQX RTOS* (document MQXGSRTOS) for details about the generic build process and compile time configuration. This chapter focuses on the steps related to makefiles only.

2.1.1 Batch Build (Windows)

MQX library build

To build libraries, launch a batch file which is in the "build/<board>/make" directory.

build/twrk60n512/make/build_gcc_arm.bat

MQX application build

To build applications, execute the batch file in the application project directory. Navigate to this directory:

/build/make/<project_name>_<board>/build_<tool>.bat

2.1.2 Batch Build (Linux)

MQX library build

To build all libraries, execute a shell script file, which is in the "build/<board>/make" directory.

build/twrk60n512/make/build_gcc_arm.sh

MQX application build

To build applications, execute the shell script file in the application project directory. Navigate to this directory:

/build/make/<project_name>_<board>/build_<tool>.sh

2.2 Makefile build

MQX library build

Navigate to the "<project/directory>/build/<project_name>_<board>/make" directory, for example "mqx/build/make/bsp_twrk60n512," and run this command to build a specific MQX library (in this case BSP in debug configuration).

Windows

mingw32-make TOOL=gcc_arm CONFIG=debug build

Linux

make TOOL=gcc_arm CONFIG=debug build

Make parameters description:

TOOL - name of the toolchain.

- **CONFIG** name of the configuration. A configuration is defined by specific flags and include paths. Configurations "**debug**" (low optimization level) and "**release**" (high optimization level) are provided.
- target Allowed targets are

	build	- run build process
	clean	- run clean process
	rebuild	- run clean and build processes
	help	- default target, prints the help
debugme	- print the int	ernal variables

MQX application build

To build or clean an application, navigate to the example directory, run "make" command and specify CONFIG, TOOL and LOAD or LINKER_FILE parameters.

In this directory:

mqx/examples/hello/build/make/hello_twrk60n512

Run this command:

Windows

mingw32-make TOOL=gcc_arm CONFIG=debug LOAD=intflash build

Linux

make TOOL=gcc_arm CONFIG=debug LOAD=intflash build

Make parameters description:

TOOL	- name of the toolchain.
CONFIG	 name of the configuration. A configuration is defined by specific flags and include paths. Configurations "debug" (low optimization level) and "release" (high optimization level) are provided.
LOAD	- name of linker file. Use this parameter when a linker command file is placed in the BSP output directory. Otherwise, use the LINKER_FILE parameter with full path.
LINKER_FILE	- full path to the linker file, use when you want to configure a full path to the linker command file and when you do not want to use a default path from BSP output directory.
APPLICATION_DIR	 output elf file location. By default, this location is /mqx/examples/<example_name>/build/make/<example_name>/gcc_arm/<ta rget>/. To change the output directory, specify the APPLICATION_DIR or the APPLICATION_FILE variable.</ta </example_name></example_name>

3 Using MQX Makefiles with Eclipse

This section describes the integration between the MQX makefiles and the Eclipse CDT development environment in the Windows operating system. However, the process is similar for the Linux operating system.

1. Create a new C Project.

New Project		
elect a wizar	4	
Create a new C	project	
Wizards:		
type filter text		
🔺 🗁 Genera		
😂 Pro	ject	
⊿ 🔁 C/C++		
C P	roject	
C+	+ Project	
📬 Ma	kefile Project with Existing Code	
🔺 🗁 CVS		
	jects from CVS	
a 🗁 RPM		
	M Project	
🔺 🗁 Tracing	10 1257 15 1-1	
	cing Project	

Figure-1 Creating a new project

2. Select the project name and place it in appropriate directory, for example: /mqx/examples/hello/build/make/hello_twrk60n512/

 C Project C Project A Directory with specified name already exist 	s.
Project name: hello_twrk60m512 Use default location Location(repo\mqx\examples\hello\build\ Choose file system: default •	gcc_arm\hello_twrk60n512 Browse
Project type:	Toolchains: Other Toolchain Cross GCC GNU Autotools Toolchain Microsoft Visual C++ MinGW GCC
Show project types and toolchains only if	they are supported on the platform

Figure-2 Naming the project

3. Set up a make utility, for example mingw32-make. Build a directory path to the makefile directory, for example, /mqx/examples/hello/build/make/hello_twrk60n512/.

You might want to change the configuration name in Manage Configurations.

type filter text	C/C++ Build	🦕 🕶 🗘 🕶
type filter text by Resource Builders C/C++ Build Build Variables Environment Logging Settings Tool Chain Editor C/C++ General Git Project References	C/C++ Build Configuration: Default [Active] Builder Settings Behaviour Refresh Policy Builder Builder type: External builder Use default build command	↓ ↓ ↓ ↓ Manage Configuration
Run/Debug Settings > Task Repository WikiText	Build location Build directory: E\git-repos\mqx-repo\mqx\examples\hello	Variables riable Refs in Makefiles s\build\make\hello_twrk60n512 Workspace File system Variables

Figure-3 Setting up make utility and a build directory

 Open a makefile batch file, for example Windows operating system: /mqx/examples/hello/build/make/hello_twrk60n512/buid_gcc_arm.bat Linux: /mqx/examples/hello/build/make/hello_twrk60n512/buid_gcc_arm.sh

4	@echo OFF	
	set NOPAUSE=%1	
	mingw32-make TOOL=gcc_arm CONFIG=debug LOAD=intflash build)-
	if errorlevel 1 (1
	set NOPAUSE=0	
í.	pause	

Figure-4 Opening makefile batch file

5. Set up commands to the textbox field "build" and "clean".

ype filter text	C/C++ Build		🗢 🗣 🗢
 Resource Builders C/C++ Build Build Variables Environment Logging 	Configuration: Default [Active]	Manage (Refresh Policy	Configuration
Settings Tool Chain Editor ▷ C/C++ General Git Project References Run/Debug Settings	Build settings	Enable parallel build Use optimal jobs (4) Use parallel jobs: Use unlimited jobs	
Task Repository WikiText	Workbench Build Behavior Workbench build type:	Make build target:	
	Build on resource save (Auto build)	all	Variables
	Note: See Workbench automatic build	preference	
	Build (Incremental build)	(TOOL=gcc_arm CONFIG=debug LOAD=intflash build)	Variables
	Clean	(TOOL=gcc_arm CONFIG=debug LOAD=intflash clean)	Variables
		Restore Defaults	Apply

Figure-5 Setting up commands for build and clean

6. Now you can build and clean the project.

2 2	Import Export	
	Build Project	
	Clean Project	
68	Refresh	F5
	Close Project	

Figure-6 Build and Clean the project

7. In the Windows operating system, if you want to see project source files, copy and replace your ".project" file with ".project" file from the "cw10gcc/hello_twrk60n512" directory. In the Linux operating system create the project content manually. Pre-defined Eclipse projects will be available in the future MQX software versions.

Output elf file is in this location:

/mqx/examples/hello/build/make/hello_twrk60n512/gcc_arm/intflash_debug/

If you want to change the output directory of the elf file, specify the APPLICATION_DIR or the APPLICATION_FILE variable during step 5.

For example, APPLICATION_DIR=../../gcc_arm/hello_twrk60n512.

You may want to import the generated binary to debug it, as described in the next chapter. To do so, go to the File/Import menu and select the C/C++ Executable under the C/C++ category.

Freescale Semiconductor

4 Running and Debugging the MQX Application Using J-Link Gdbserver

This description applies to the TWR-K20D50M BSP, the Hello World example, and the J-Link for the ARM[®] hardware debug probe. The same procedure also applies to all other BSPs and examples.

4.1 Starting the gdbserver - Windows

The gdbserver is a bridge between the GDB, the GNU project debugger, and the J-Link debug probe. The gdbserver executable is in this folder:

<jlink_tools_install_dir>/JLinkGDBServer.exe

1. When you run the executable, the connection dialog appears where you can select the connection options for your board.

SEGGER J-Link GDB Server V4.76e - Config
Connection to J-Link
👁 USB 🔲 Serial No.
O TCP/IP
Target device
MK20DX128xxx5
Little endian 🖉
Target interface
JTAG 👤
Speed
O Auto selection
C Adaptive clocking
Command line option
-select USB -device MK20DX128xxx5 -if JTAG -speed 1000
OK Cancel

Figure-7 J-Link

2. After selecting the options for the board, press the OK button and the dialog indicating the Gdbserver status and the configuration details, such as the listening port, appears.

SEGGER J-Link GDB Server V4.76e		X		
File Help				
J-Link Connected	itial JTAG speed 1000 kHz ent JTAG speed 1000 kHz	✓ Localhost only Stay on top ✓ Show log window Generate logfile ✓ Verify download ✓ Init regs on start		
Log output: Clear log				
SEGGER J-Link GDB Server V4.70	je	*		
JLinkARM.dll V4.76e (DLL comp:	iled Sep 20 2013 16:06	5:28)		
<pre>SEGGER J-Link GDB Server V4.76e JLinkARM.dll V4.76e (DLL compiled Sep 20 2013 16:06:28)GDB Server start settings GDBInit file: none GDB Server Listening port: 2331 SWO raw output listening port: 2332 Terminal I/O port: 2332 Terminal I/O port: 2333 Connecting to J-Link J-Link is connected. Firmware: J-Link ARM V8 compiled Jul 17 2013 11:24:15 Hardware: V8.00 S/N: OEM: IAR Checking target voltage Listening on TCP/IP port 2331 Connecting to target J-Link found 1 JTAG device, Total IRLen = 4 JTAG ID: 0x4BA00477 (Cortex-M4) Connected to target Waiting for GDB connection</pre>				
0 Bytes downloaded 1 JTA	G device			

Figure-8 Gdbserver listening port

3. After the gdbserver is running, you are ready to connect to it. You may debug the application either from the command line or using Eclipse with CDT. Note that before you try debugging, you may want to download the application to the target device.

4.2 Starting the gdbserver - Linux

The gdbserver for Linux is available only in a command line version. To start the J-Link gdbserver, use this command:

JLinkGDBServer -select USB -device MK60DN512Zxxx10 -if JTAG -speed auto

- The "device" option specifies the silicon name and uses silicon-specific settings.
- The "if" option specifies the debug interface. Valid values are JTAG, SWD.

See Section BSP to CPU name mapping for J-Link gdbserver for valid device values and supported BSPs.

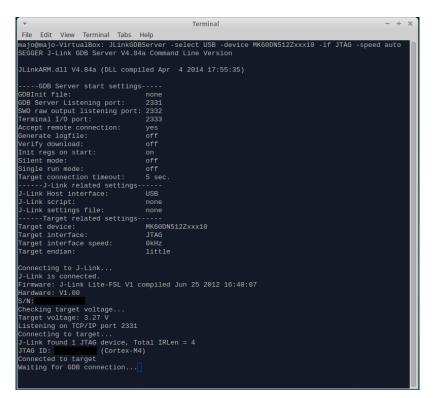


Figure-9 Terminal

The gdbserver is listening on port 2331. After the gdberver runs, you are ready to connect to it. You may debug the application either from the command line or using Eclipse with CDT. See next chapter for more information.

4.3 Using Eclipse with the gdbserver

After the gdbserver is running, you can download the firmware to the board and start debugging. These chapters describe how to use the command line GDB client and the GDB client Eclipse integration.

4.3.1 Using the command line interface

Follow these steps to download the application to your evaluation board and start debugging.

• Run the J-Link gdbserver with valid options, for example:

JLinkGDBServer -select USB -device MK60DN512Zxxx10 -if JTAG -speed auto

- Run the arm-none-eabi-gdb -tui hello_twrk60n512.elf where:
 - o --tui options switch GDB to curses view
 - o hello twrk60n512.elf represent the path to the elf with debug symbols
- Attach the GDB client to the gdbserver by: target remote localhost:2331
- Load your application to the board using the load hello_twrk60n512.elf, where hello_twrk60n512.elf represents the path to your application.
- Perform the monitor reset command to reset the chip and prevent registers from getting default values.
- Set a breakpoint, for example break main or break hello task.
- Run the application with the continue command.

Using the command line interface for debugging works for intflash build targets of the Kinetis processor family.

Debugging a Vybrid processor is more complex and the basic initialization can be found here:

mqx/source/bsp/[BOARD]/gcc_arm/jlinkgdb/[BOARD].dgbinit

4.3.2 Creating the debug configuration in Eclipse

Users should install the Eclipse with CDT and the C/C++ GDB Hardware Debugging feature. The instructions apply for the Windows operating system. However, a similar approach should work on a Linux Host.

 Go to the "Run/Debug Configurations..." menu and create a new GDB Hardware Debugging configuration. First, modify the Launcher and select the "Standard GDB Hardware Debugging Launcher" instead of the default "GDB (DSF) Hardware Debugging Launcher". See the figure.

Debug Configurations			23
Create, manage, and run con	figurations		Ť
Yppe filter text € C/C++ Application € C/C++ Attach to Applin E C/C++ Postmortem De C C/C++ Remote Application E C/C++ Remote Application Image: Second S	Name: hello Main Debugger > S C/C++ Application: hello.elf Project: hello Build (if required) before launc Build configuration: Enable auto build Use workspace settings	Startup 15/2 Source Common Variables Search Project hing Use Active Verification using 'C/C++ Application' Disable auto build Configure Workspace Settings	Browse Browse
 III → Filter matched 20 of 20 items 	Using Standard GDB Hardware I other	Debugging Launcher - <u>Select</u> Apply	Revert
?		Debug	Close

Figure-10 Selecting the launcher

2. Next, in the Debugger tab, set the path to the GDB client from the GNU Tools package and uncheck the "Use remote target" option as shown in the image.

Debug Configurations	X
reate, manage, and run configur	ations
Yepe filter text Cyce+ Application C/C++ Application C/C++ Remote Application D/Java Applet Java Applet Java Applet Java Applet J/JUHIR Bug- in Test Leuch Group Maven Build OSG Framework RAP Application SAP App	Name: hello Main (* Debugger > Startup * Source Common GD8 Setup GD8 Command: C.\Program Files\GNU Tools ARM Embedded(4.7.2013q3\bin\arm-none-eabi-gdb.exe Erowse) Variables Command Set: Standard (Windows) Protocol Version: mi Verbose console mode Remote Target Use remote target JTAG Device: Generic TCP/IP Host name or IP address: localhost Port number: 10000
 III ► Filter matched 20 of 20 items 	Using Standard GDB Hardware Debugging Launcher - <u>Select other</u> Apply Revert
?	Debug

Figure-11 Selecting the path to the GDB client

3. Set the startup options by going to the "Startup" tab and pasting this content into the "Initialization Commands" text field:

```
# connect to the gdb server
target remote localhost:2331
# Set gdb server to little endian
monitor endian little
# Set JTAG speed to 1000 kHz
monitor speed 1000
# Reset the target
monitor reset
monitor sleep 100
# Set JTAG speed in khz
monitor speed auto
# Vector table placed in RAM
monitor writeu32 0xE000ED08 = 0x1FFF8000
```

Note: Update the gdbserver listening port. You may find it in the gdbserver information dialog – see the Starting the gdbserver sections. In this case the port number is 2331.

4. Paste this content into the "Run Commands" text field:

```
monitor reg r13 = (0x1FFF8000)
monitor reg pc = (0x1FFF8004)
```

5. In the "Runtime Options" set the initial breakpoint at the main function and mark the "Resume" option.

The final result should look like this figure.

reate, manage, and run configu	ations
Very Filter teat C/C++ Atach to Application C/C++ Postmontem Debugging C/C++ Remote Application COB Hardware Debugging C/D+ Remote Application SoB Hardware Debugging Diva Applet Java Jav	Name: hello Main % Debugger > Statup % Source Common Initialization Commands Reset and Delay (seconds): 3 V Hat Connect to the gdb sever target remote localhost2331 S st gdb server to little endian montor endian little Load Image and Symbols V Load image
	(e) Use project binay: hello.elf Use file: Workspace Image offset (hex): Image offset (hex): Use project binay: hello.elf Use file: Use file: Workspace File System Symbols offset (hex): Runtime Options Step rogram counter at (hex): Ist program counter at (hex):
	Wine Run Commands monitor reg rl 3 = (0x1FFF8000) * wonktor reg pc = (0x1FFF8004) * Using Standard GDB Hardware Debugging Launcher - Select other Apply

Figure-12 Complete setup

6. Click on the "Apply" button and then the "Debug" button. The gdbserver application should flash the microcontroller (download the firmware to the target) with your application and the debugger should stop at the main function.

7. The gdbserver application indicates the successful connection with the green light next to the "GDB" text field. It also contains the log output, which might be helpful if there are issues.

SEGGER J-Link GDB Server V4.76e	
File Help	
GDB 1 client @ 127.0.0.1 Initial JTAG speed 1000 kHz 💌	 Localhost only Stay on top
J-Link Connected Current JTAG speed 4000 kHz	Show log window
CPU MK20DX128xxx5, Halted 3.28 V	 Verify download Init regs on start
Log output: Clear log	
Downloading 16272 bytes @ address 0x00008300 Downloading 14616 bytes @ address 0x0000C290 Downloading & bytes @ address 0x0000FB80 Downloading 4 bytes @ address 0x0000FB80 Downloading 96 bytes @ address 0x0000FB88 Downloading 924 bytes @ address 0x0000FC18 Writing register (PC = 0x000006FC (Data = 0x490E480 Read 2 bytes @ address 0x00004354 (Data = 0xF64F) Read 2 bytes @ address 0x00004356 (Data = 0xF020) Read 2 bytes @ address 0x00004356 (Data = 0xF64F) Read 2 bytes @ address 0x00004356 (Data = 0xF64F) Read 2 bytes @ address 0x00004356 (Data = 0xF64F) Read 2 bytes @ address 0x00004354 (Data = 0xF64F) Setting breakpoint @ address 0x00004354, Size = 2, Starting target CPU Breakpoint reached @ address 0x00004354	
	•
63 KB downloaded 1 JTAG device	1.

Figure-13 Successful connection

4.3.3 BSP to CPU name mapping for J-Link gdbserver

If using the Segger J-Link gdbserver to download firmware to the board, select the correct flashing algorithm. See this table.

Board	CPU ID	Board	CPU ID
kwikstikk40x256	MK40DX256xxx10	twrk60d100m	MK60DN512xxx10
twrk20d50m	MK20DX128xxx5	twrk60f120m	MK60FN1M0xxx12
twrk20d72m	MK20DX256xxx10	twrk60n512	MK60DN512xxx10
twrk21d50m	MK21DN512xxx5	twrk70f120m	MK70FN1M0xxx12
twrk21f120m	MK21FN1M0xxx12	twrvf65gs10_a5	VF6xx_A5
twrk40d100m	MK40DX256xxx10	twrvf65gs10_m4	VF6xx_M4
twrk40x256	MK40DX256xxx10	vybrid_autoevb_a5	VF6xx_A5
twrk53n512	MK53DN512xxx10	vybrid_autoevb_m4	VF6xx_M4

Table-1 Flashing algorithm

5 Makefiles Structure

This is the color key in this document this to express dependencies.

mqx/bsp- is a directory of library or applicationbsp- is a name of library or applicationtwrk60n512- is board nameqcc arm- is a toolchain name

5.1.1 Makefiles hierarchy

Library build process consists of partial makefiles: /mqx/bsp/build/make/bsp_twrk60n512/Makefile /mqx/bsp/build/make/bsp_twrk60n512/tools/gcc_arm.mak /build/common/make/global.mak /build/twrk60n512/make/tools/gcc_arm.mak /build/common/make/verify.mak /build/common/make/lib-process.mak

Application build process consists of partial makefiles: /mqx/examples/hello/build/make/hello_ twrk60n512/Makefile /mqx/examples/hello/build/make/hello_ twrk60n512/tools/gcc_arm.mak /build/common/make/global.mak /build/twrk60n512/make/tools/gcc_arm.mak /build/common/make/verify.mak /build/common/make/app-process.mak

5.1.2 Partial makefiles definition

/mqx/bsp/build/make/bsp_twrk60n512/Makefile

This makefile sets up common SOURCES, INCLUDE paths and mandatory variables:

MQX_ROOTDIR– path to mqx root directoryTYPE– type of build, setup to "library" valueNAME– library nameBOARD– name of the board

Using Eclipse and GDB with Freescale MQX™ RTOS User's Guide, Rev. 1, 08/2014

Freescale Semiconductor

LIBRARY_ROOTDIR - rootdir of libraries built for specific board and tool		
LIBRARY_DIR	 path to library output directory 	
LIBRARY_FILE	 path to library output file 	
POSTBUILD_CMD	- macro to obtain post build command. Depends on HOSTENV	

/mqx/bsp/build/make/bsp_twrk60n512/tools/gcc_arm.mak

This makefile sets up tool chain-specific SOURCES and INCLUDE paths.

5.1.3 Partial application makefiles

/mqx/examples/hello/build/make/hello_twrk60n512/Makefile

This makefile sets up common SOURCES, INCLUDE paths and mandatory variables:

MQX_ROOTDIR	 path to mqx root directory
TYPE	- "application" value
NAME	- application name
BOARD	– name of board
LIBRARY_ROOTDIR	- rootdir of libraries builded for specific board and tool
APPLICATION_DIR	 path to application output directory
APPLICATION_FILE	 path to library output file
LINKER_FILE	 macro to obtain linker command file

/mqx/bsp/build/make/bsp_twrk60n512/tools/gcc_arm.mak

This makefile sets up toolchain-specific SOURCES and INCLUDE paths.

5.1.4 Partial common makefiles

/build/common/make/global.mak

This partial makefile contains common macros, default definitions of TOOLCHAIN_ROOTDIR, and HOSTENV variables.

/build/twrk60n512/make/tools/gcc_arm.mak

This partial makefile contains sub-paths to toolchain binaries, common flags, and definitions.

/build/common/make/verify.mak

This partial makefile performs existence verification of linker command files, toolchain paths, and valid command line variables.

/build/common/make/app-process.mak

This partial makefile contains targets, rules, and a dependency to build an application.

/build/common/make/lib-process.mak

This partial makefile contains targets, rules, and a dependency to build a library.