STMicroelectronics

Nomadik toolset getting started

User manual

8087207 Rev A

January 2008

BLANK

User manual

Nomadik toolset getting started

Nomadik is a registered trademark of STMicroelectronics

The Nomadik family of multimedia application processors are targeted at the mobile phone and portable multimedia markets. Nomadik processors are multicore processors, consisting of a single processing core (an ARM core) and a number of audio and video accelerator cores (also known as media processor cores). Applications intended for the Nomadik processor can take full advantage of the multi-core architecture by distributing their functionality between all the available cores, thereby gaining advantages in terms of speed of execution, small memory footprint and power consumption.

This manual provides a brief overview to the process of developing firmware for Nomadik processors, and also introduces the concepts the Nomadik Multiprocessor Framework (NMF), which is provided specifically for designing and developing distributed, multimedia applications for the Nomadik processor.

Contents

Preface

Comments on this manual should be made by contacting your local STMicroelectronics sales office or distributor.

Nomadik Toolset documentation suite

The Nomadik Toolset documentation suite comprises the following volumes:

Nomadik Toolset getting started

ADCS 8087207. This manual describes the principles of the Nomadik Toolset and provides some basic worked examples.

Nomadik MMDSP+ Toolset

ADCS 8086787. This manual describes the Nomadik MMDSP+ Tools for compiling, debugging and simulating MMDSP code on Nomadik cores.

NMF programming model

ADCS 8071313. This manual describes the Nomadik multimedia framework (NMF) programming model that is used to define the Nomadik component-based projects.

NOMADIK toolset system trace

ADCS 8101318. This document briefly describes how to operate the system trace mechanism within the Nomadik Toolset environment.

Nomadik debugging configuration

ADCS 8105391. This document describes the configurations supported by the toolset for the various debuggers that can connect to the Nomadik boards.

ST ARM GDB debugger

ADCS 7833754. This manual provides information about starmgdb debugger specific commands and connection. It provides the ARM semi-hosting support for hardware target, OS21 thread awareness and a command for monitoring the component status of NMF applications

Trace32 for Nomadik

ADCS 8063903. This manual describes how to use Trace32 to configure and connect to a target board through a PowerTrace or PowerDebug box.

armgcc interoperability with RVDS/RVCT

ADCS 8108043. This manual describes how to write source code that can be compiled by both the armgcc and the ARM RVCT compiler, as well as how to mix object modules coming from both compilers into the same application.

GNU ARM

This is the full set of Gnu documents describing how to compile, assemble, link debug and profile applications on the ARM core using the standard GNU tools.

Conventions used in this guide

General notation

The notation in this document uses the following conventions:

- `sample code`, `keyboard input` and `file names`,
- *variables*, *`code variables`* and *`code comments`*,
- `equations` and `math`,
- **screens**, **windows**, **dialog boxes** and **tool names**,
- **instructions**.

Software notation

Syntax definitions are presented in a modified Backus-Naur Form (BNF) unless otherwise specified.

- Terminal strings of the language, that is those not built up by rules of the language, are printed in teletype font. For example, `void`.
- Nonterminal strings of the language, that is those built up by rules of the language, are printed in italic teletype font. For example, *`name`*.
- If a nonterminal string of the language starts with a nonitalicized part, it is equivalent to the same nonterminal string without that nonitalicized part. For example, `vspace-`*`name`*.
- Each phrase definition is built up using a double colon and an equals sign to separate the two sides (‘`::=`’).
- Alternatives are separated by vertical bars (‘`|`’).
- Optional sequences are enclosed in square brackets (‘`[`’ and ‘`]`’).
- Items which may be repeated appear in braces (‘`{`’ and ‘`}`’).

Acknowledgements

ARM®, Multi-ICE® and RealView® are registered trademarks of ARM limited in the EU and other countries.

Windows® is a registered trademarks of Microsoft Corporation in the United States and/or other countries.

Linux® is a registered trademark of Linus Torvalds.

Trace32® is a registered trademark of Lauterbach Datentechnik GmbH.

Terminology

ST Micro Connect

The first ST Micro Connect product was named the “ST Micro Connect”, it is now known as the “ST Micro Connect 1” and the term “ST Micro Connect” is used to refer to the family of ST Micro Connect devices. The latest product is the “ST Micro Connect 2”. In some instances these names are abbreviated to “STMC”, “STMC1” and “STMC2”.

CheapTap

CheapTap is an ST development box that is not for widespread distribution.

1 Introduction

The Nomadik Toolset is an integrated collection of compilers, linkers, debuggers and other related software for developing and debugging multimedia firmware and software for all available configurations of the Nomadik multimedia application processor. Because the Nomadik has multiple cores with different architectures, a specific set of tools is required to develop software components for each of the different types of core.

The tools are as follows:

- ARM GNU code generation and debug tools (for the main processor core)
- ARM RVDS code generation tools (also for the main processor core)
- MMDSP+ code generation, debugging, simulating and profiling tools (for the media processor cores)

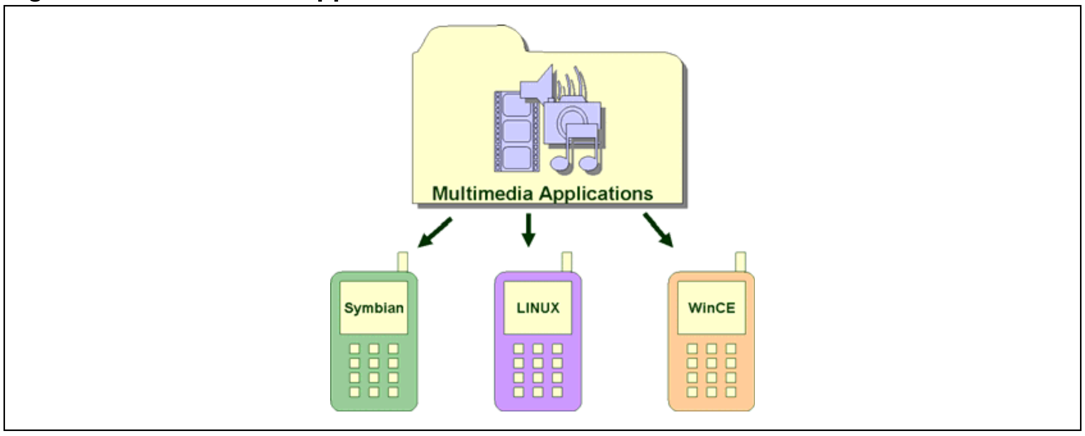
Note: Only one from the ARM GNU compiler tools and the ARM RVDS compiler tools are required for developing applications, although both sets may be provided as part of the Nomadik Toolset. The ARM GNU compiler tools are the default toolset.

The toolset includes the Nomadik Multiprocessor Framework (NMF) tools for designing and developing Nomadik applications that take full advantage of the multi-core architecture.

The toolset also includes the STWorkbench integrated development environment, based upon the Eclipse open development platform, see *www.eclipse.org*.

Nomadik supports a wide range of operating systems; these include Symbian, Linux and WinCE, see *Figure 1*.

Figure 1. Multimedia applications



The Nomadik Toolset software runs independently of the final operating system and is complementary to, and inter-operable with, existing OS aware tools.

The NMF provides a framework for addressing the problem of distributing the software code execution intelligently between the ARM and MPC cores. The underlying concept behind NMF is the component, which is a self-contained software unit designed to carry out a particular task. Components communicate with one another using interfaces, which means a completed application is essentially a network of components, all linked together by their various interfaces.

The Nomadik toolkit has tools for developing and debugging components, tuning their performances, and finally integrating them together into a final application.

2 Development flow for the multimedia application

This chapter describes the various stages of software development using the Nomadik Toolset.

The top-down approach for developing multimedia applications using NMF is as follows.

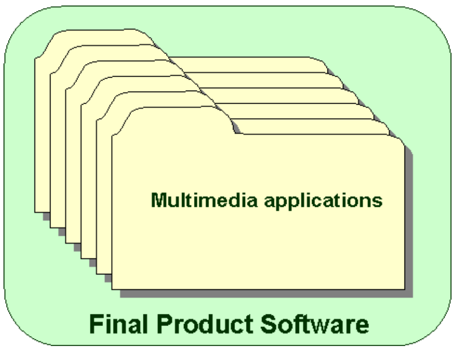
1. Product software design. Identify the multimedia applications that are required for the final product, see *Section 2.1: Product software design*.
2. Multimedia application architecture design. Identify the individual components that are to be networked together to create each of the final multimedia applications identified at step 1, see *Section 2.2: Multimedia application architecture design*.
3. Multimedia application development and test environment. Develop and test the components identified at step 2 outside of the final OS context, see *Section 2.3: Multimedia application development and test environment*.
4. Integrate the components that were developed and tested at step *3* into a multimedia application in the target operating system, see *Section 2.4: Integration of the multimedia application in the final OS*.

2.1 Product software design

A product that uses a Nomadik processor requires a set of applications to provide the multimedia functionality that a consumer expects. Such applications typically provide a combination of audio, video and imaging features.

At this high level, the application design does not need to be tied to any underlying operating system, see *Figure 2*.

Figure 2. Final product software



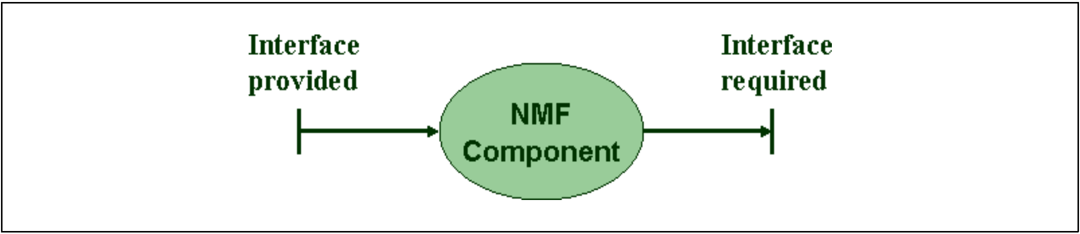
2.2 Multimedia application architecture design

A distributed, multimedia application running on Nomadik comprises a number of individual components. Each component is dedicated to carrying out a specific task, and each component communicates with others through its interfaces. NMF provides two low-level software applications, the “component manager”, and the “execution engine” that schedules the components, allocates resources to and facilitates communication between the components and the underlying operating system.

To construct the finished application, these self-contained components are the “building blocks” that link together by means of their interfaces. Components and their interfaces should be designed to be as flexible as possible so that any given component can be reused in different contexts and be shared between different applications.

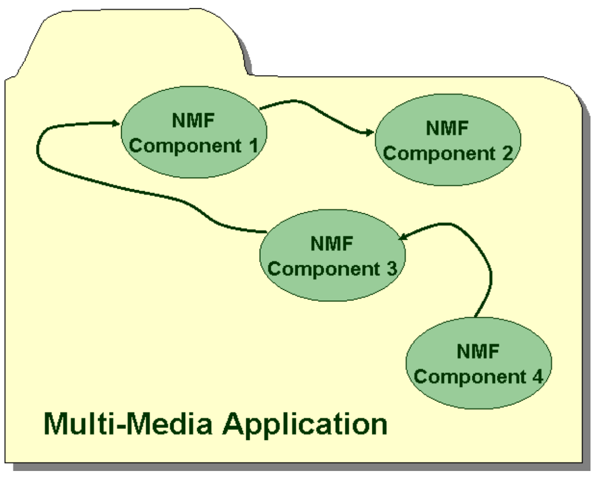
When an interface is implemented for a component, it fulfils one of two different roles, either as a **provided interface** (that is, it provides a service to other components) or as a **required interface** (that is, it requires a service from another component). *Figure 3* shows the provided and required interfaces of a component.

Figure 3. NMF components



Each provided interface is “bound” to the required interface of another component. The binding of interfaces is supervised by the underlying NMF software and requires only minimum involvement from the developer. The result of this interface binding is a multimedia application constructed from a network of components, see *Figure 4*.

Figure 4. Multimedia application



The principles of NMF are explained in more detail in the *Nomadik multiprocessing framework programming model (ADCS 8071313)*.

2.3 Multimedia application development and test environment

The development and initial testing of a multimedia application is done on a PC running Microsoft Windows or RHEL Linux. This phase is carried out under the STWorkbench development environment using either the MPC core simulator that is provided as part of the Nomadik Toolset, or the connection to real silicon. Initially, components can be developed and tested individually and then linked together (using their interfaces) to be tested in context.

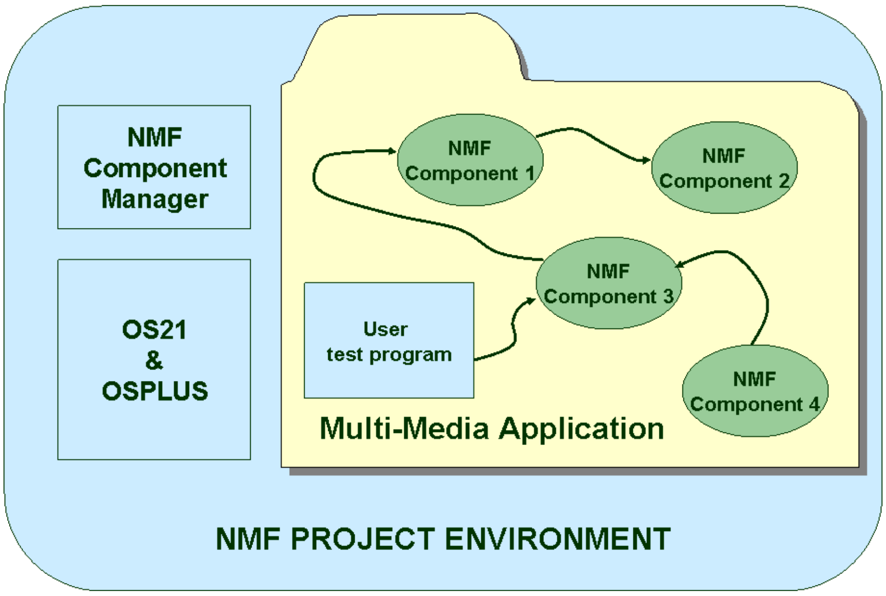
A demonstration or a test program for a multimedia application can be developed entirely within the PC environment, for example, to investigate the performance of the application design. Fine tuning the balance between components' execution can be undertaken as part of this testing phrase, see *Figure 5*.

When the application satisfactorily performs on the PC, the application can be migrated to a test board that features a Nomadik processor with the same configuration as the final target. Final testing and adjustments are carried out as part of this phase.

The Nomadik Toolset provides a range of tools for these phases of the development process. The following lists some of these tools.

- C code generators and debug tools for the various Nomadik cores.
- A set of debug tools for the various Nomadik cores.
- OS21 is a real-time OS environment for the ARM core that provides thread support, interrupt services and other basic features.
- OSPlus is an extension of OS21 for system integration that provides high speed file-system access on external devices present on the development boards.
- The NMF Component Manager, integrated with OS21, that provides the services for instantiating and starting component execution.
- The NMF Execution Engine is a lightweight type of scheduler that runs on Nomadik MPC cores.
- Multi-core debug tools for both Nomadik silicon platforms (boards) and Nomadik virtual platforms (SoC simulators) that include OS21 and NMF-aware features.
- System level views that provide system information about the application distribution among the Nomadik cores.
- System profiling tools that offer dynamic information on the multimedia application executing on the Nomadik cores.
- Platform utilities for system-level configuration, flashing and security tools.

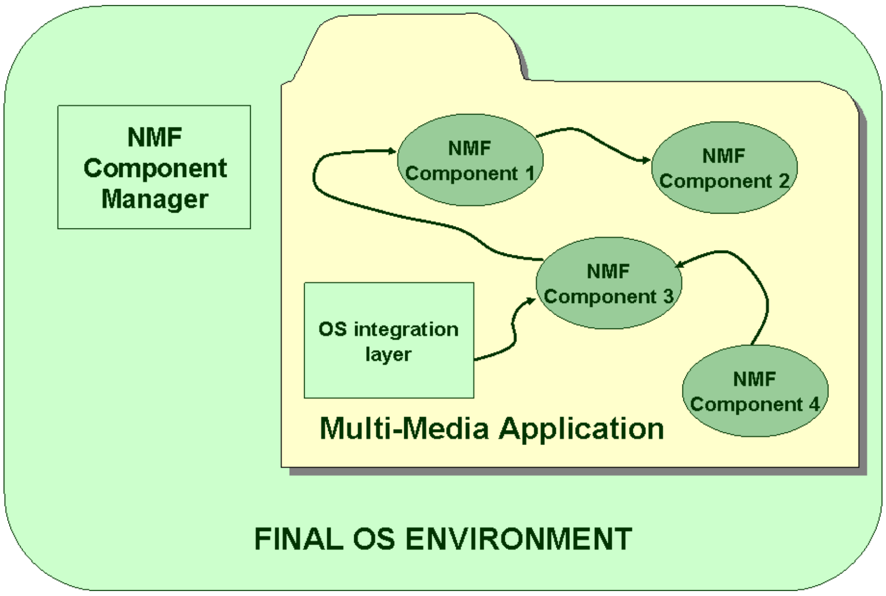
Figure 5. Multimedia application development and test environment



2.4 Integration of the multimedia application in the final OS

The Nomadik Toolset ensures the portability of multimedia applications between an NMF development project to the final application running under the final operating system. This is achieved through the use of an NMF Component Manager implementation, which is specific to the final operating system, see *Figure 6*. NMF Components that have been created using an NMF project do not have to be recompiled and can be transferred directly to the final operating environment.

Figure 6. Integration of the multimedia application in the final OS



3 Working with an NMF Project

This chapter briefly describes how to create and manage an NMF project using the STWorkbench integrated development environment. For a full explanation of the MMDSP+ Toolset, see the *Nomadik MMDSP+ toolset user manual (ADCS 8086797)*.

3.1 What is an NMF project?

All the source codes and related files required for the development of a multimedia application using NMF are organized into a single NMF project. The project can be maintained from the STWorkbench or by using command line tools.

All the files related to a project are held in sub-directories of the project directory. The project directory has the same name as the project.

An NMF project is typically subdivided into a number of parts. One part of the project contains the code for the software targeted to the ARM core (the ARM area) and the other component parts contain the code for the software that is targeted to the various MPC cores (the component area).

A set of coding guidelines and compiler options are provided to ensure the interoperability of the Nomadik Toolset with third party tools like ARM RVDS or Lauterbach, and with code already developed with the ARM RVDS tools.

An NMF project can be created for the following two kind of test environments.

- The NMF-BARE test environment, which integrates a minimal support of execution environment for the code running on ARM core.
- The NMF-OS21 test environment, which integrates the access to the OS21 services for the test program running on ARM.

The NMF test environment has no impact on the binary NMF components produced by the NMF project. All the examples described in this document refer to the NMF-OS21 test environment.

3.2 How to manage an NMF project

We recommend that managing NMF projects is through a set of makefiles that can be called either from the command line, or from STWorkbench through a **Makefile project**. A set of example makefiles are delivered in the `<install_dir>/nmf/examples` directory.

3.2.1 Building makefiles for an NMF project

The NMF project makefiles call the following tools.

- The NMF command line level tools.
- The ARM command line level tools.

  The ARM tools are either the ARM RVDS or ARM GNU tools. The ARM RVDS tools are described in documentation provided by ARM.

  The ARM GNU command line tools are described in the following documents:

  – GNU C preprocessor and compiler manuals
  – GNU Assembler
  – GNU Linker
  – GNU Binutils
  – GNU Profiler
  – GNU Debugger

  The *armgcc interoperability with RVDS/RVCT user manual (ADCS 8108043)* gives information on how to use ARM RVDS and ARM GNU compilers together.
- The MMDSP+ command line level tools, see the *MMDSP+ toolset user manual (ADCS 8071313)*.

Makefiles can be directly executed from the Nomadik Toolset command prompt environment on Windows. The Nomadik Toolset provides the `make` tool, as well as the minimal Unix-like utility tools which are a subset of MSYS. See `<install_dir>/bin/msys/1.0/bin` for a full list of these tools.

Note: No specific support is provided on Linux.

All examples described in this document are based on the Windows platform.

There is no specific rule to follow for writing makefiles for NMF projects. However, we recommend that a new NMF project makefile is derived from one of the examples delivered in the `<install_dir>/nmf/examples` directory. These examples handle many target variants (such as the Nomadik chip version, the ARM code generation tools selection, and so on) that is possibly useful to manage in the project.

Build example

To start the command line demo for the “helloworld” example in the command prompt, type:

```
cd %NDKTOOLS%/nmf/examples/helloworld
make NMF_ENVIRONMENT=os21 NMF_TOOLCHAIN=gnu
```

Type `make help` to get more information on the current configuration supported by the example.

3.2.2 Managing an NMF project using STWorkbench

To launch STWorkbench, either:

- from the Windows **Start** menu, select **Programs > Nomadik Toolset x.y.z > Nomadik STWorkbench**, or
- type `nomadik_workbench` at the Nomadik command prompt

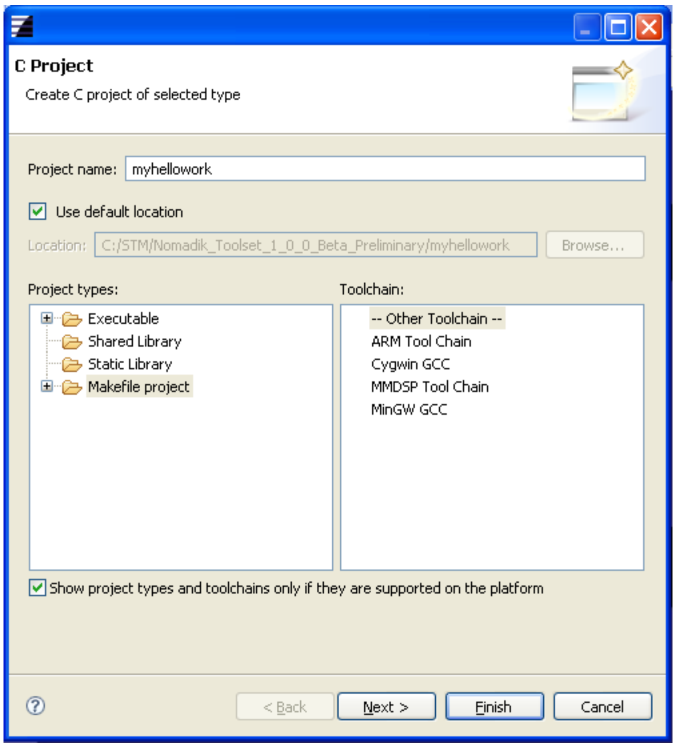
On first use of STWorkbench, you are required to enter the location of the workspace. The workspace is the root location of all project files.

Caution: On Windows, do not accept the default location of the workspace, which is proposed under the “Documents and Settings” area. The reason for this is that several Nomadik tools do not support blank characters.

To create a project, follow the steps below.

1. Select **File > New > C Project**. The **C Project** dialog box opens, see *Figure 7*.

Figure 7. C Project dialog box



C Project
Create C project of selected type

Project name: myhellowork

☑ Use default location

Location: C:/STM/Nomadik_Toolset_1_0_0_Beta_Preliminary/myhellowork

Project types:
- Executable
- Shared Library
- Static Library
- Makefile project

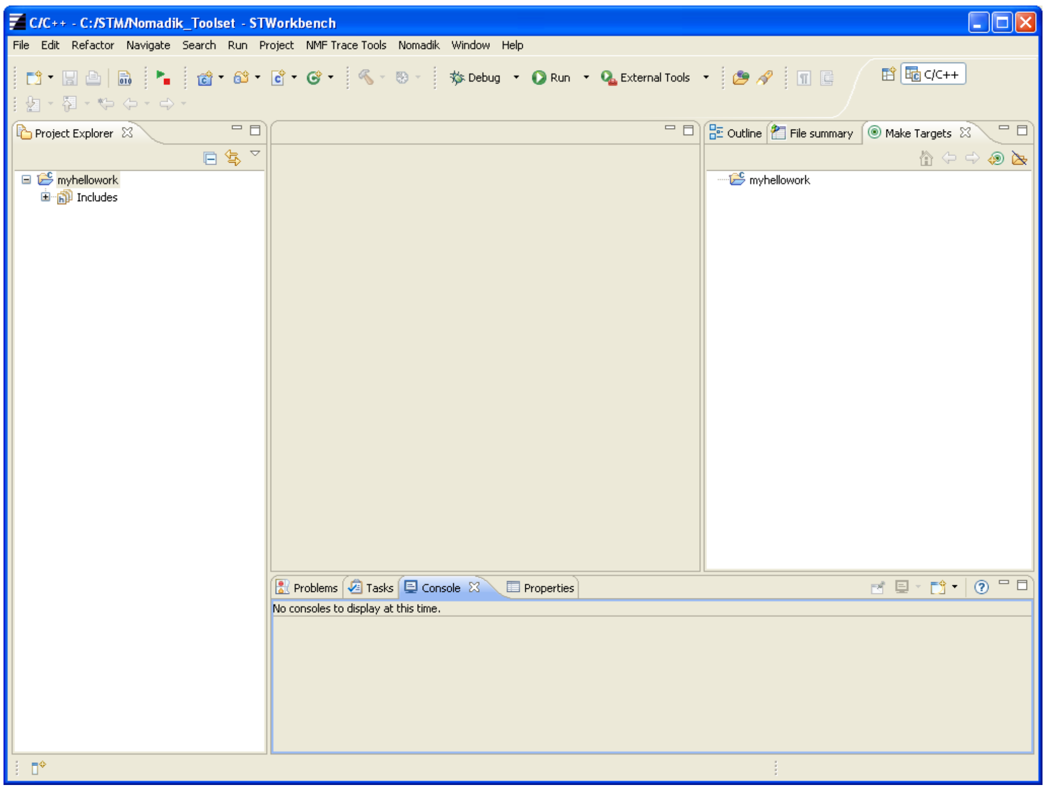
Toolchain:
- -- Other Toolchain --
- ARM Tool Chain
- Cygwin GCC
- MMDSP Tool Chain
- MinGW GCC

☑ Show project types and toolchains only if they are supported on the platform

2. Type a **Project name**, for example `myhellowork`.

3. Select **Makefile project** and click the **Finish** button.
   STWorkbench creates and opens the project, see *Figure 8*.

Figure 8. Hello World example

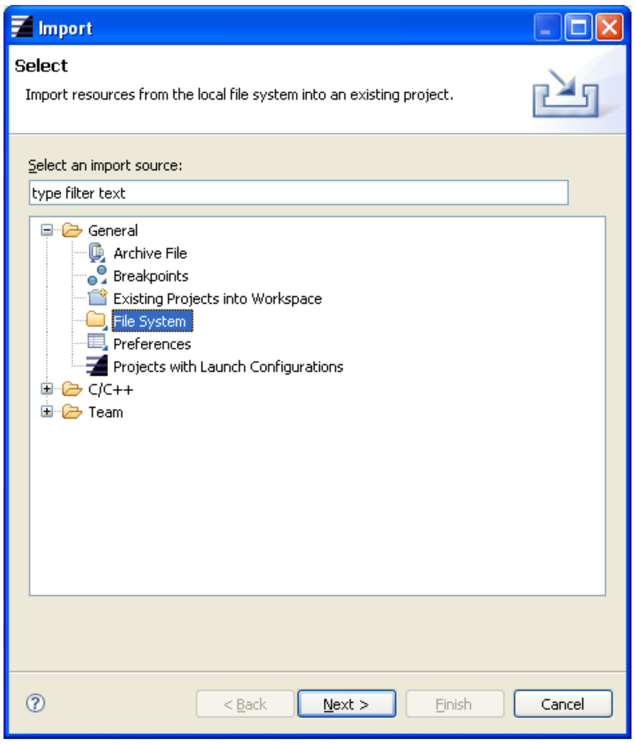


No consoles to display at this time.

The next step is to create the project files (sources, makefiles and so on) under this project. You may also decide to import the full set of files of an existing project. To do this, follow the steps below.

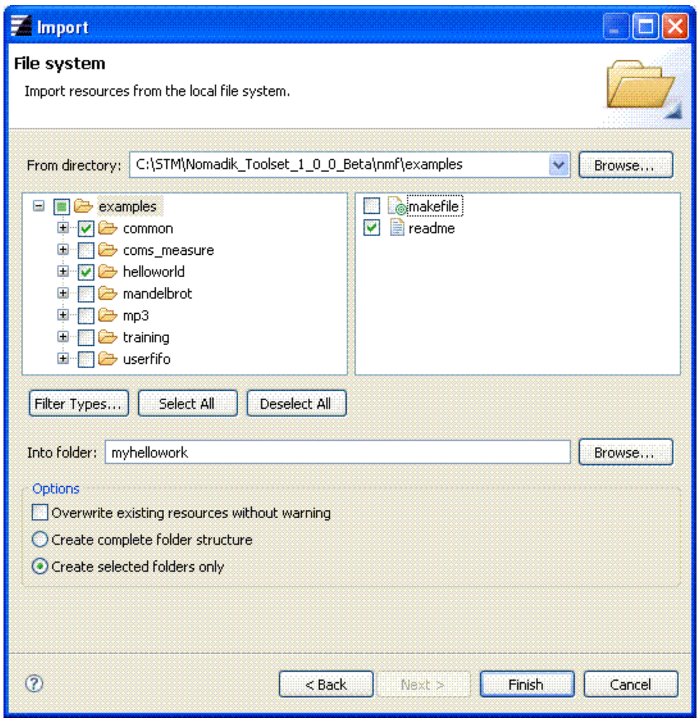
1. Select **File > Import**. the **Import** dialog box opens, see *Figure 9*.

Figure 9. Import window, Select



2. Select the **File System** source and click on **Next**. The **File system** panel of the **Import** window opens, see *Figure 10*.

Figure 10. Import window, File system


Import

File system

Import resources from the local file system.

From directory: C:\STM\Nomadik_Toolset_1_0_0_Beta\nmf\examples

examples
[✓] common
[ ] coms_measure
[✓] helloworld
[ ] mandelbrot
[ ] mp3
[ ] training
[ ] userfifo

[ ] makefile
[✓] readme

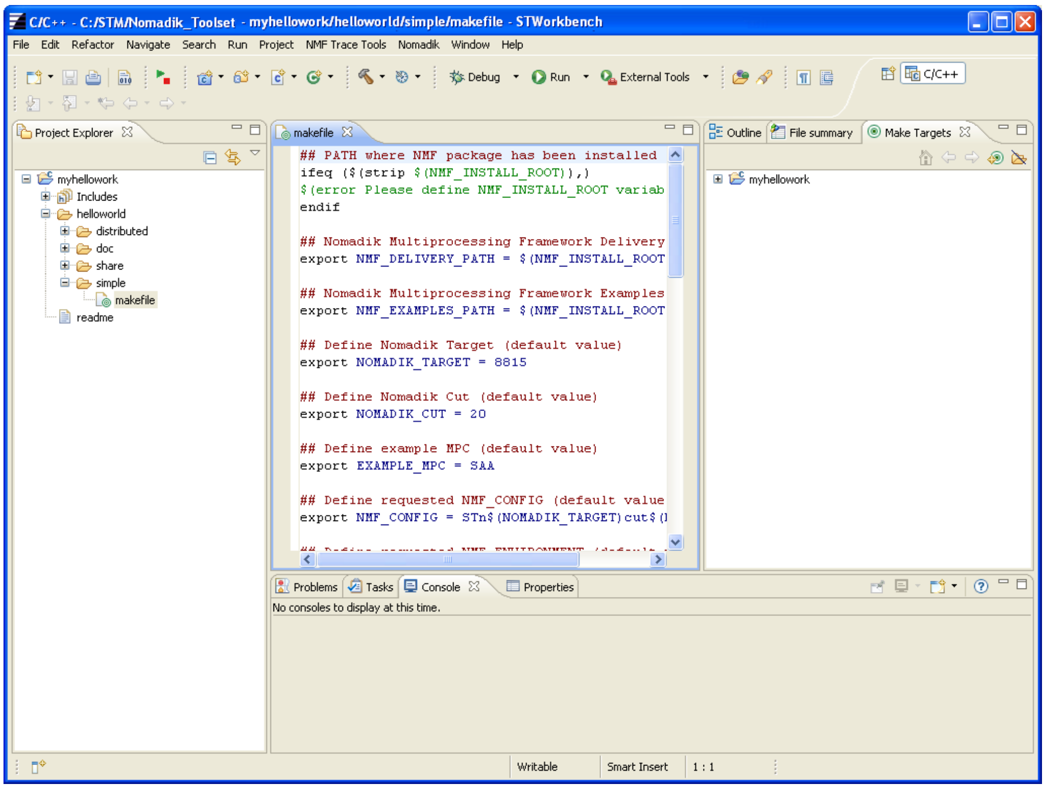
Into folder: myhellowork

Options
[ ] Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

3. In the **From directory** field, select the directory containing the existing project. In this case, `<install_dir>\nmf\examples`.
4. Expand the **examples** directory and select the **common** directory, the **helloworld** project and the **readme** file.
5. In the **Into folder** field select the **myhellowork** folder.
6. Click the **Finish** button.

After the import is completed, you can browse the source files of the NMF project, and start to modify them, see *Figure 11*.

Figure 11. Example makefile


C/C++ - C:/STM/Nomadik_Toolset - myhellowork/helloworld/simple/makefile - STWorkbench
File Edit Refactor Navigate Search Run Project NMF Trace Tools Nomadik Window Help
Debug Run External Tools C/C++
Project Explorer
myhellowork
Includes
helloworld
distributed
doc
share
simple
makefile
readme
makefile
Outline File summary Make Targets
myhellowork
Problems Tasks Console Properties
Writable Smart Insert 1 : 1

```
## PATH where NMF package has been installed
ifeq ($(strip $(NMF_INSTALL_ROOT)),)
$(error Please define NMF_INSTALL_ROOT variab
endif

## Nomadik Multiprocessing Framework Delivery
export NMF_DELIVERY_PATH = $(NMF_INSTALL_ROOT

## Nomadik Multiprocessing Framework Examples
export NMF_EXAMPLES_PATH = $(NMF_INSTALL_ROOT

## Define Nomadik Target (default value)
export NOMADIK_TARGET = 8815

## Define Nomadik Cut (default value)
export NOMADIK_CUT = 20

## Define example MPC (default value)
export EXAMPLE_MPC = SAA

## Define requested NMF_CONFIG (default value
export NMF_CONFIG = STn$(NOMADIK_TARGET)cut$(I
```

No consoles to display at this time.

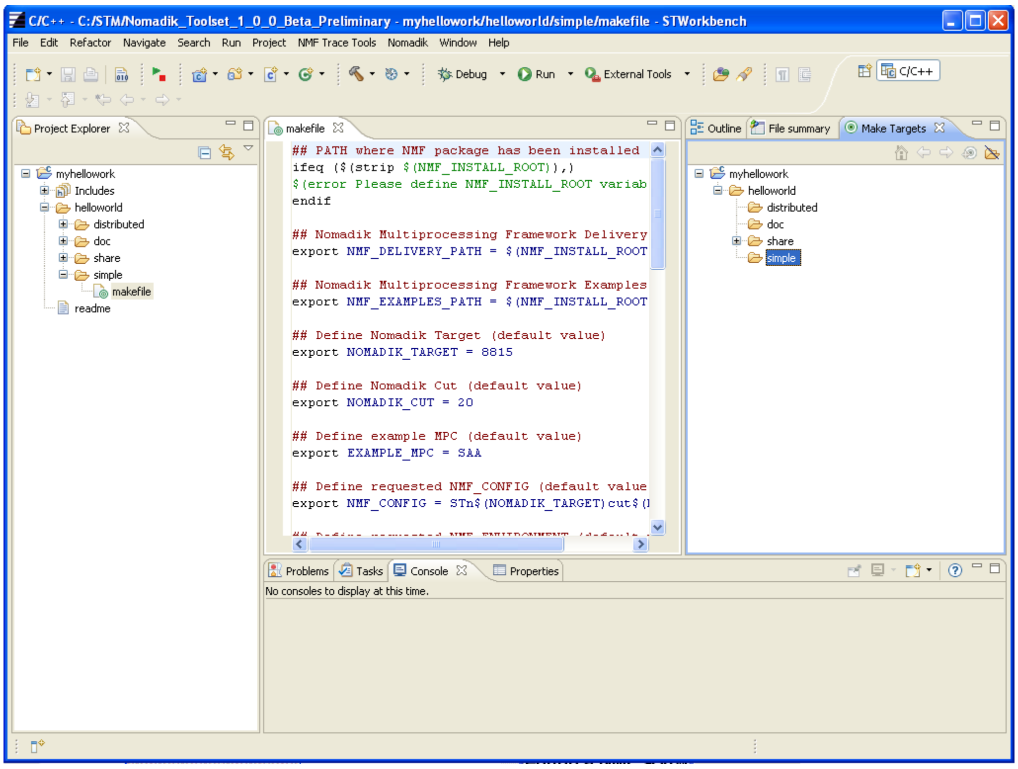
Note: *To make this example makefile work, replace the* `NMF_EXAMPLES_PATH` *definition with the full directory name to the project files in the current workspace. For example:*

`export NMF_EXAMPLES_PATH = C:/temp/workspace-beta-p1/myhellowork`

To build this NMF helloworld simple project, first create a dedicated Make target. To do this, follow the steps below.

1. In the **Make Targets** tab (in the right panel), select the **simple**, see *Figure 12*.

Figure 12. Make Targets tab

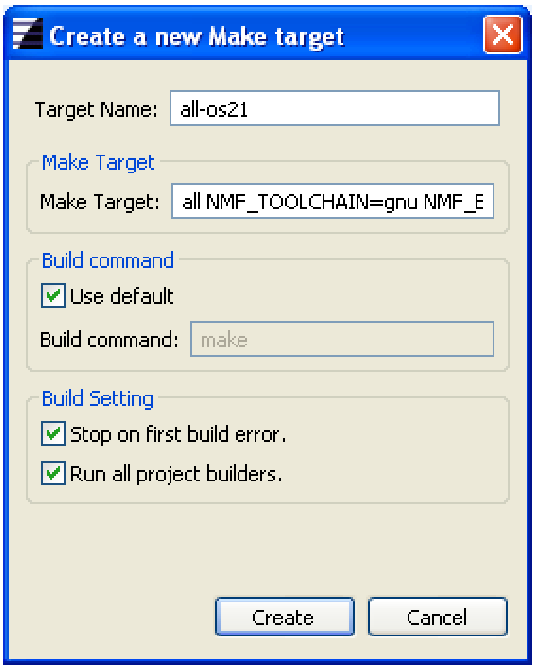


```
## PATH where NMF package has been installed
ifeq ($(strip $(NMF_INSTALL_ROOT)),)
$(error Please define NMF_INSTALL_ROOT variab
endif

## Nomadik Multiprocessing Framework Delivery
export NMF_DELIVERY_PATH = $(NMF_INSTALL_ROOT

## Nomadik Multiprocessing Framework Examples
export NMF_EXAMPLES_PATH = $(NMF_INSTALL_ROOT

## Define Nomadik Target (default value)
export NOMADIK_TARGET = 8815

## Define Nomadik Cut (default value)
export NOMADIK_CUT = 20

## Define example MPC (default value)
export EXAMPLE_MPC = SAA

## Define requested NMF_CONFIG (default value
export NMF_CONFIG = STn$(NOMADIK_TARGET)cut$(
```

No consoles to display at this time.

2. Right-click on **simple** and select **Add Make Target**. The Make Target dialog box opens, see *Figure 13*.

Figure 13. Create a new Make target


Create a new Make target

Target Name: all-os21

Make Target

Make Target: all NMF_TOOLCHAIN=gnu NMF_E

Build command

✓ Use default

Build command: make

Build Setting

✓ Stop on first build error.

✓ Run all project builders.

3. Type a **Target Name** (for example, `all-os21`) and type the following to select the ARM GNU toolchain and the OS21 test environment.

```
all NMF_TOOLCHAIN=gnu NMF_ENVIRONMENT=os21
```

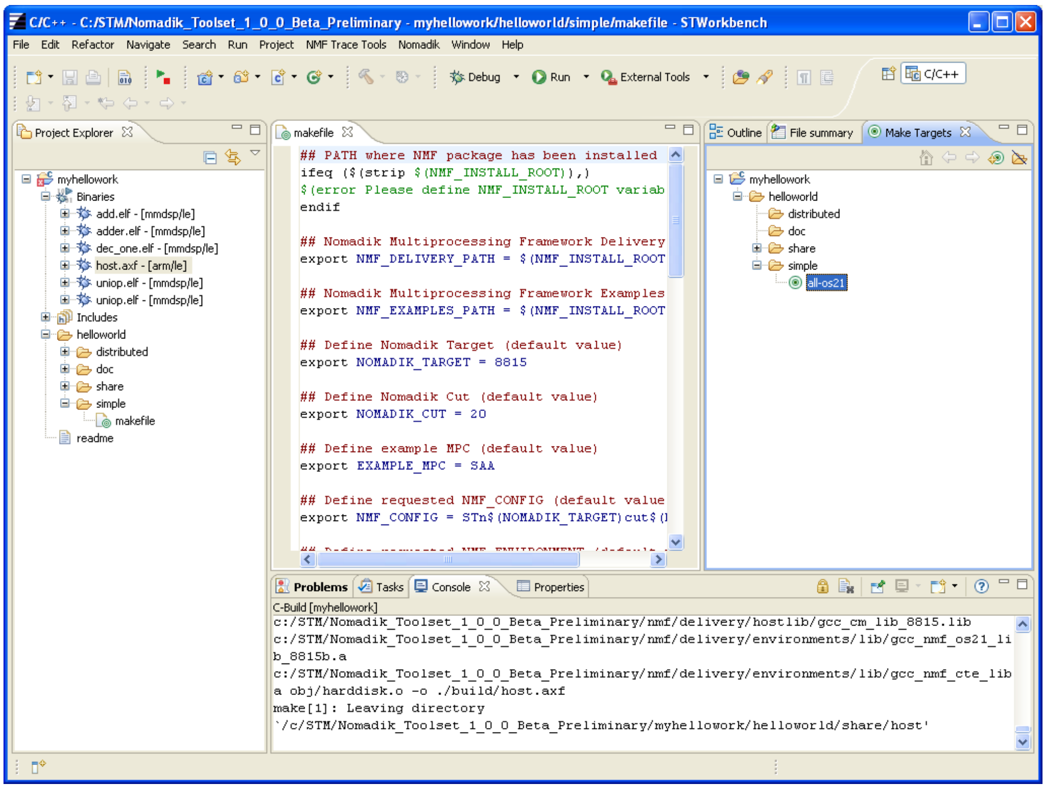
4. Click the **Create** button.

Note: The readme file delivered in the NMF sample directory provides more information on the parameters of the make files.

To build the project, expand the **simple** folder and double-click the **all-os21** Make Target.

The **Console** tab at the bottom of the window displays the output of the build. When the build is complete, the binary file `host.axf` is visible in the project file list, see *Figure 14*.

Figure 14. Completed build



```
## PATH where NMF package has been installed
ifeq ($(strip $(NMF_INSTALL_ROOT)),)
$(error Please define NMF_INSTALL_ROOT variab
endif

## Nomadik Multiprocessing Framework Delivery
export NMF_DELIVERY_PATH = $(NMF_INSTALL_ROOT

## Nomadik Multiprocessing Framework Examples
export NMF_EXAMPLES_PATH = $(NMF_INSTALL_ROOT

## Define Nomadik Target (default value)
export NOMADIK_TARGET = 8815

## Define Nomadik Cut (default value)
export NOMADIK_CUT = 20

## Define example MPC (default value)
export EXAMPLE_MPC = SAA

## Define requested NMF_CONFIG (default value
export NMF_CONFIG = STn$(NOMADIK_TARGET)cut$(I
```

```
c:/STM/Nomadik_Toolset_1_0_0_Beta_Preliminary/nmf/delivery/hostlib/gcc_cm_lib_8815.lib
c:/STM/Nomadik_Toolset_1_0_0_Beta_Preliminary/nmf/delivery/environments/lib/gcc_nmf_os21_li
b_8815b.a
c:/STM/Nomadik_Toolset_1_0_0_Beta_Preliminary/nmf/delivery/environments/lib/gcc_nmf_cte_lib
a obj/harddisk.o -o ./build/host.axf
make[1]: Leaving directory
`/c/STM/Nomadik_Toolset_1_0_0_Beta_Preliminary/myhellowork/helloworld/share/host'
```

3.3 Designing NMF software architecture

Designing an application that uses the NMF software architecture requires some knowledge of the NMF methodology. See the *NMF programming model user manual (ADCS 8071313)* for more information.

The objective of the architecture design phase is to analyze the functionality of the application and determine the basic components that are required to provide that functionality. The next step is to determine, from the list of components, the interfaces that are required between the components. One of the principles of NMF is that the definition of the interfaces is kept separate from the implementation of the components themselves.

The interfaces are specified using an NMF specific language called IDL (interface definition language.)

At the architecture design phase, the decision as to which MPC core is to be used for any given component has not yet been made.

3.4 Implementing an NMF software application

The Implementation phase references two areas within the project, the ARM area and the Component area.

3.4.1 ARM area

The implementation process requires that a test program or demonstrator is implemented to run on the ARM core. The purpose of this test program is primarily to deal with the instantiation of each component on the Media Processor cores of the Nomadik.

The test program has to be implemented following the API described in the online documentation *NMF Components Manager API*.

To implement programs for NMF projects to run on the ARM core, there are two sets of tools:

- ARM GNU compiler tools
- ARM RVDS compiler tools

An NMF project can target the following two test environments.

- The NMF-BARE test environment, which integrates a minimal support of execution environment for the code running on ARM core.
- The NMF-OS21 test environment, which integrates the access to the OS21 services for the test program running on ARM.

OS21 is described in the *OS21 User Manual* (ADCS 7358306) and *OS21 for ARM User Manual* (ADCS 8083358)

Note: *To benefit from the OS21 thread-safe calls to the C standard library, the usage of the GNU ARM standard library (newlib) is mandatory.*

In this case, the RVDS code generator can still be used providing that the interoperability rules are followed, see the armgcc interoperability with RVDS/RVCT user manual (ADCS 8108043).

3.4.2 Component area

Each primitive component is implemented (using the C language) as a stand-alone entity. Each component must strictly follow the definition of its specified interfaces.

The online documentation *NMF Generic MPC API* describes how to import the interfaces required and how to export the interfaces provided by the component.

Note: In this version of the Nomadik Toolset, the components must run on an MPC core and therefore require to be compiled with the MMDSP+ Compiler.

3.5 Debugging an NMF application in a multi-core environment

Because an NMF application is distributed across a number of different cores, it is possible that the application requires several different debuggers to run concurrently.

The Nomadik Toolset provides two solutions, these are:

- homogeneous, multi-core debug solutions
  - **STWorkbench** provides a solution that is based on the standard Eclipse and GNU gdb solutions that have been enhanced with OS21 and NMF aware debug features, see *Section 3.5.1*.
  - **Trace32** third party debugger provided by Lauterbach provides also a multi-core integrated solution for Nomadik, with NMF aware features.

    Details on how to use the Trace32 debugger on Nomadik boards are described in the *Trace32 for Nomadik user manual (ADCS 8063903)*.
- heterogeneous multi-core debug solutions

  This is provided by using one ARM debugger (either **starmgdb** or any third party debugger) for the ARM area of the NMF application, and one debugger for each MPC core for debugging the components. The MPC debugger must be NMF aware. The MPC debuggers that are available are:
  - the **mmgdb** debugger provided in the Nomadik Toolset
  - the **mmdsp-Trace32** third party debugger provided by Lauterbach

  Multi-core debugging with the ARM **rvdebug** debugger and the **mmgdb** debugger is described in the *Nomadik Debugging Configuration user manual (ADCS 8105391)*.

.

3.5.1 Debugging an NMF application with STWorkbench

This section refers to debugging an NMF application with STWorkbench.

To debug a multi-core NMF application using STWorkbench, connect the **starmgdb** debugger directly to the ARM core and connect one **mmgdb** debugger for each MPC core to the Nomadik chip.

Connection boxes provide a single box, multi-core debug solution under the STWorkbench. Ensure they are plugged between the debuggers and the Nomadik silicon board. The three supported connection boxes are:

- the ARM RealView-ICE box, connected through ethernet only

Note: Only version 3.0 or later of the RealView-ICE firmware is supported. Refer to the MMDSP+ toolset user manual for more information about RealView-ICE firmware configuration.

- the ST Micro Connect box, connected through Ethernet or an USB port
- the ST Cheaptap dongle, which is a USB JTAG adaptor

To enable concurrent **starmgdb** and **mmgdb** debugger connection to the Nomadik chip, the two ST boxes require the ST Micro Connect server.

The configuration of the connection of the **starmgdb** debugger to these connection boxes is described in the *STARM Debugging tools User Manual*. Additional information related to the configuration of the connection of the mmgdb debugger to these connection boxes can be found in the *MMDSP+ toolset user manual*.

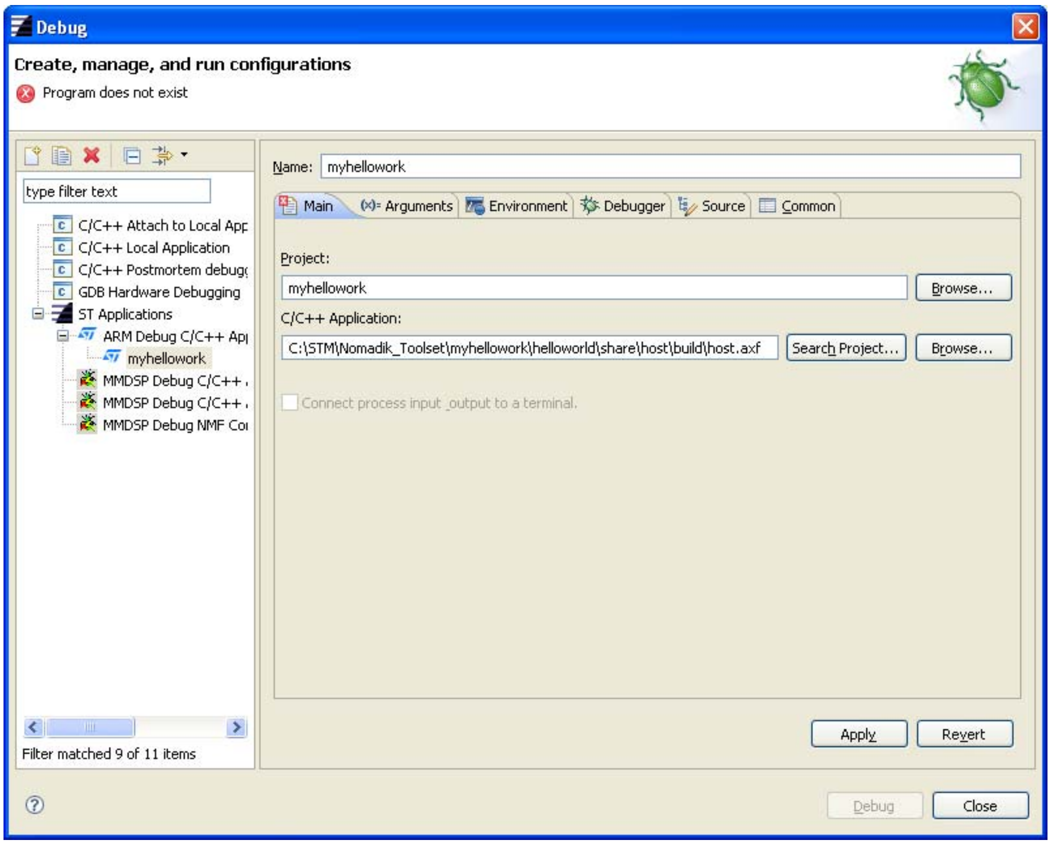
This section assumes that an ARM RealView-ICE connection box is being used, correctly set up and configured.

Configure STWorkbench to debug the ARM core

Use the following instructions to set up the debugging for the ARM core.

1. From the **Run** menu, select **Open Debug Dialog**, see *Figure 15*.

Figure 15. Debug window, Main tab



Name: myhellowork

Project: myhellowork

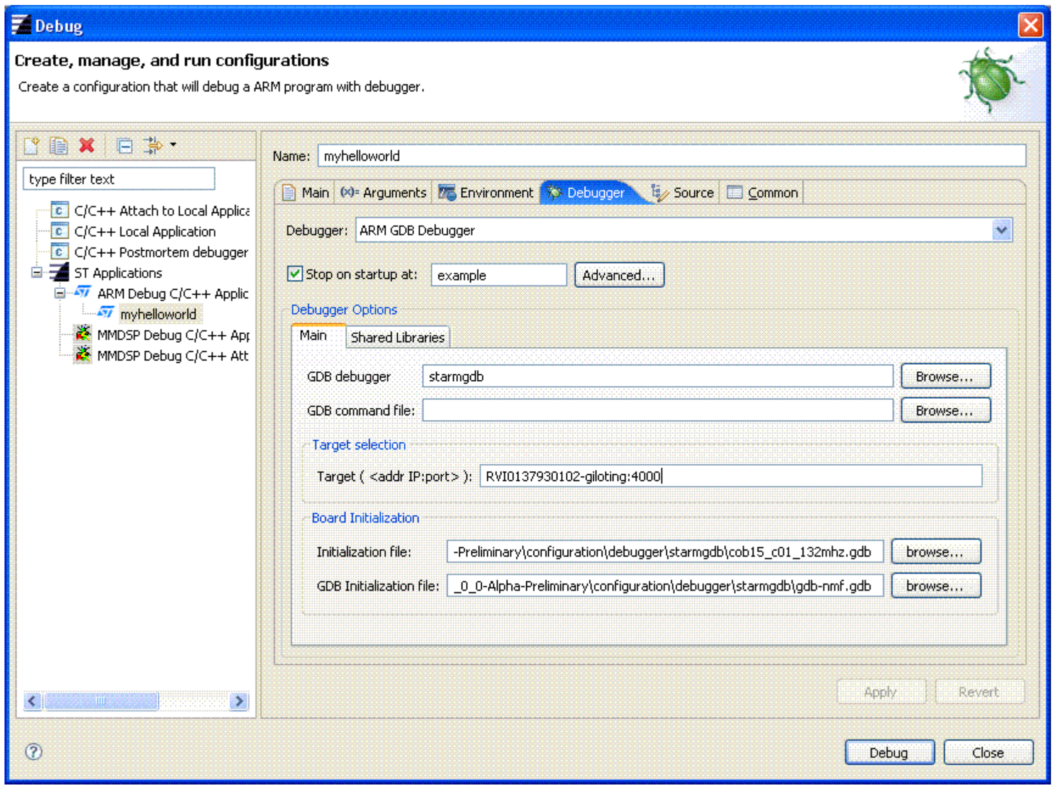
C/C++ Application: C:\STM\Nomadik_Toolset\myhellowork\helloworld\share\host\build\host.axf

Connect process input output to a terminal.

2. Expand **ST Applications**, select **ARM Debug C/C++ Application** and click the **New configuration** button.
3. In the **C/C++ Application** field, browse for the location of the `host.axf` binary file.

4. Select the **Debugger** tab, from the **Debugger** drop-down list, select **ARM GDB Debugger**, see *Figure 16*.

Figure 16. Debug window, Debugger tab



Debug

Create, manage, and run configurations

Create a configuration that will debug a ARM program with debugger.

Name: myhelloworld

Debugger: ARM GDB Debugger

[✓] Stop on startup at: example

Debugger Options

GDB debugger: starmgdb

GDB command file:

Target selection

Target (<addr IP:port>): RVI0137930102-giloting:4000

Board Initialization

Initialization file: -Preliminary\configuration\debugger\starmgdb\cob15_c01_132mhz.gdb

GDB Initialization file: _0_0-Alpha-Preliminary\configuration\debugger\starmgdb\gdb-nmf.gdb

5. Select the **Stop on startup at** checkbox and type the name of the `example` user function.
6. In the **Target selection** area, select **GDB Server on RealView Ice** and in the **Target** field, type the IP name of the RealView-ICE, followed by “:4000”. For example `RVI0154920336:4000`.
7. In the **Initialization file** field, type the path to the GDB initialization script that corresponds to the board.
8. In the **GDB initialization file** field, type the path to the `nmf-os21-ndk15.gdb` file.

Configure STWorkbench to debug an MPC core

Use the following instructions to set up the debugging for an MPC core.

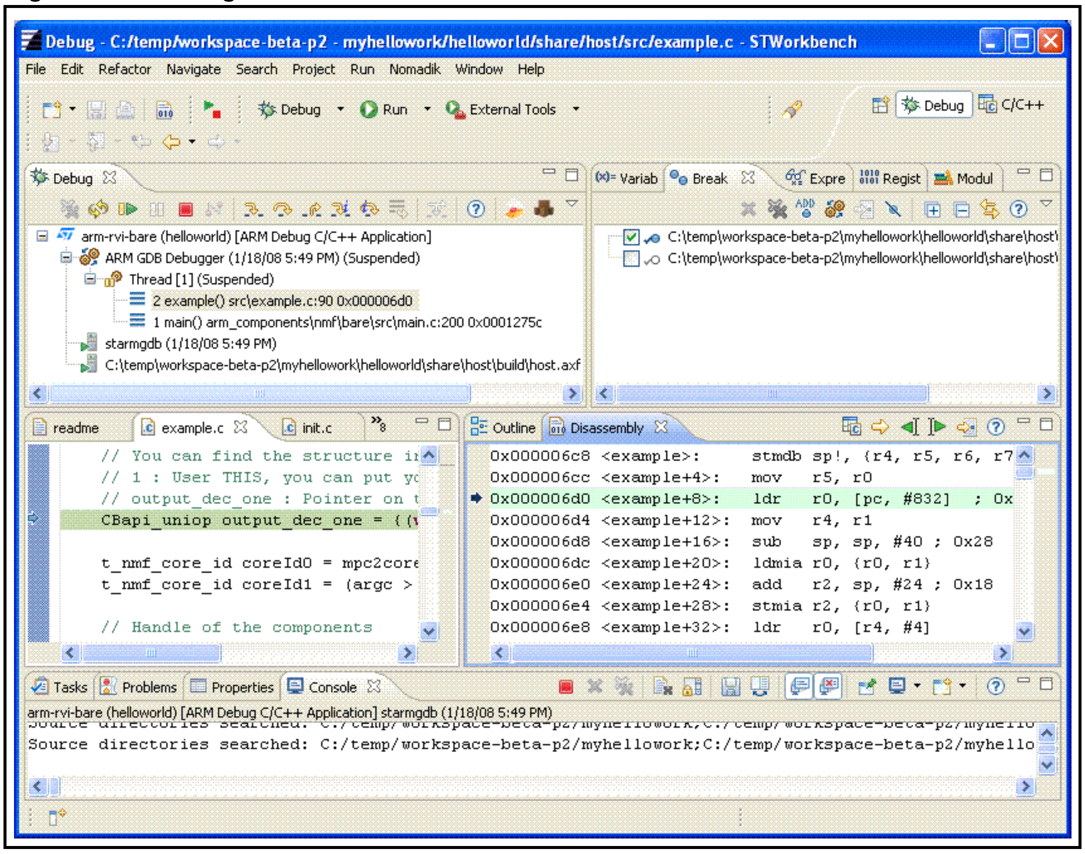
1. Select **Open Debug Dialog** from the **Run** menu.
2. Expand **ST Applications**, select **MMDSP Debug NMF Component (attach to running target)**.
3. In the **Select NMF Executable Engine** field, browse for the location of the `synchronous_8815_hsem.elf` file (`<`*`install_dir`*`>\nmf\delivery\repository\stn8815`).
4. In the **Components** area, click on **Add** and browse for the directory where the NMF components have been built.
5. Select the **Debugger** tab and from the **Debugger** drop-down list, select **MMDSP GDB Debugger**.
6. From the **Target** drop-down list, select **emu**.
7. Select the **Emulator Project** corresponding to the targeted MMDSP core.
8. Select the **Emulator Protocol** which identifies the connection to your board. These are taken from the `hw_targets.cfg` file which is described in the *MMDSP+ toolset user manual*.

Start a multi-core debug session

Use the following instructions to start a multi-core debug session.

1. Launch the ARM debug configuration previously created in *Configure STWorkbench to debug the ARM core on page 25*. The debug session stops at the user code position, see *Figure 17*.

Figure 17. Debug session


Debug - C:/temp/workspace-beta-p2 - myhellowork/helloworld/share/host/src/example.c - STWorkbench
File Edit Refactor Navigate Search Project Run Nomadik Window Help
Debug Run External Tools
Debug C/C++
Debug
Variab Break Expre Regist Modul
readme example.c init.c
Outline Disassembly
Tasks Problems Properties Console

```
arm-rvi-bare (helloworld) [ARM Debug C/C++ Application]
  ARM GDB Debugger (1/18/08 5:49 PM) (Suspended)
    Thread [1] (Suspended)
      2 example() src\example.c:90 0x000006d0
      1 main() arm_components\nmf\bare\src\main.c:200 0x0001275c
  starmgdb (1/18/08 5:49 PM)
  C:\temp\workspace-beta-p2\myhellowork\helloworld\share\host\build\host.axf
```

```
C:\temp\workspace-beta-p2\myhellowork\helloworld\share\host\
C:\temp\workspace-beta-p2\myhellowork\helloworld\share\host\
```

```
// You can find the structure in
// 1 : User THIS, you can put yo
// output_dec_one : Pointer on t
CBapi_uniop output_dec_one = { (v

t_nmf_core_id coreId0 = mpc2core
t_nmf_core_id coreId1 = (argc >

// Handle of the components
```

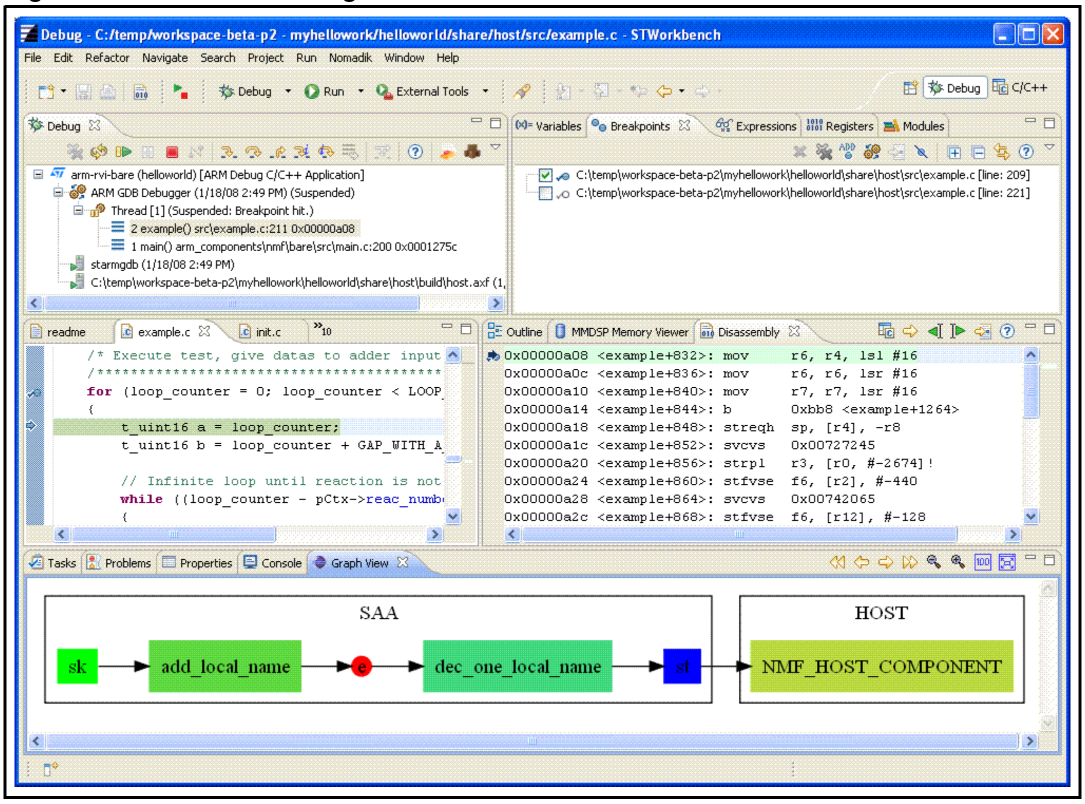
```
0x000006c8 <example>:     stmdb sp!, {r4, r5, r6, r7
0x000006cc <example+4>:   mov   r5, r0
0x000006d0 <example+8>:   ldr   r0, [pc, #832]  ; 0x
0x000006d4 <example+12>:  mov   r4, r1
0x000006d8 <example+16>:  sub   sp, sp, #40 ; 0x28
0x000006dc <example+20>:  ldmia r0, {r0, r1}
0x000006e0 <example+24>:  add   r2, sp, #24 ; 0x18
0x000006e4 <example+28>:  stmia r2, {r0, r1}
0x000006e8 <example+32>:  ldr   r0, [r4, #4]
```

```
arm-rvi-bare (helloworld) [ARM Debug C/C++ Application] starmgdb (1/18/08 5:49 PM)
Source directories searched: C:/temp/workspace-beta-p2/myhellowork;C:/temp/workspace-beta-p2/myhello
Source directories searched: C:/temp/workspace-beta-p2/myhellowork;C:/temp/workspace-beta-p2/myhello
```

2. Execute the application in the ARM debugger context to a point where the NMF components have been instantiated before starting to debug the NMF components on the MPC cores.

   At this point, you can open the view of the NMF network that graphically shows the list of the instantiated components on each MPC core with their interface binding, see *Figure 18*. In the Debug window, click the **Refresh Status** button.

Figure 18. NMF Monitoring



```
arm-rvi-bare (helloworld) [ARM Debug C/C++ Application]
  ARM GDB Debugger (1/18/08 2:49 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      2 example() src\example.c:211 0x00000a08
      1 main() arm_components\nmf\bare\src\main.c:200 0x0001275c
  starmgdb (1/18/08 2:49 PM)
  C:\temp\workspace-beta-p2\myhellowork\helloworld\share\host\build\host.axf (1,
```

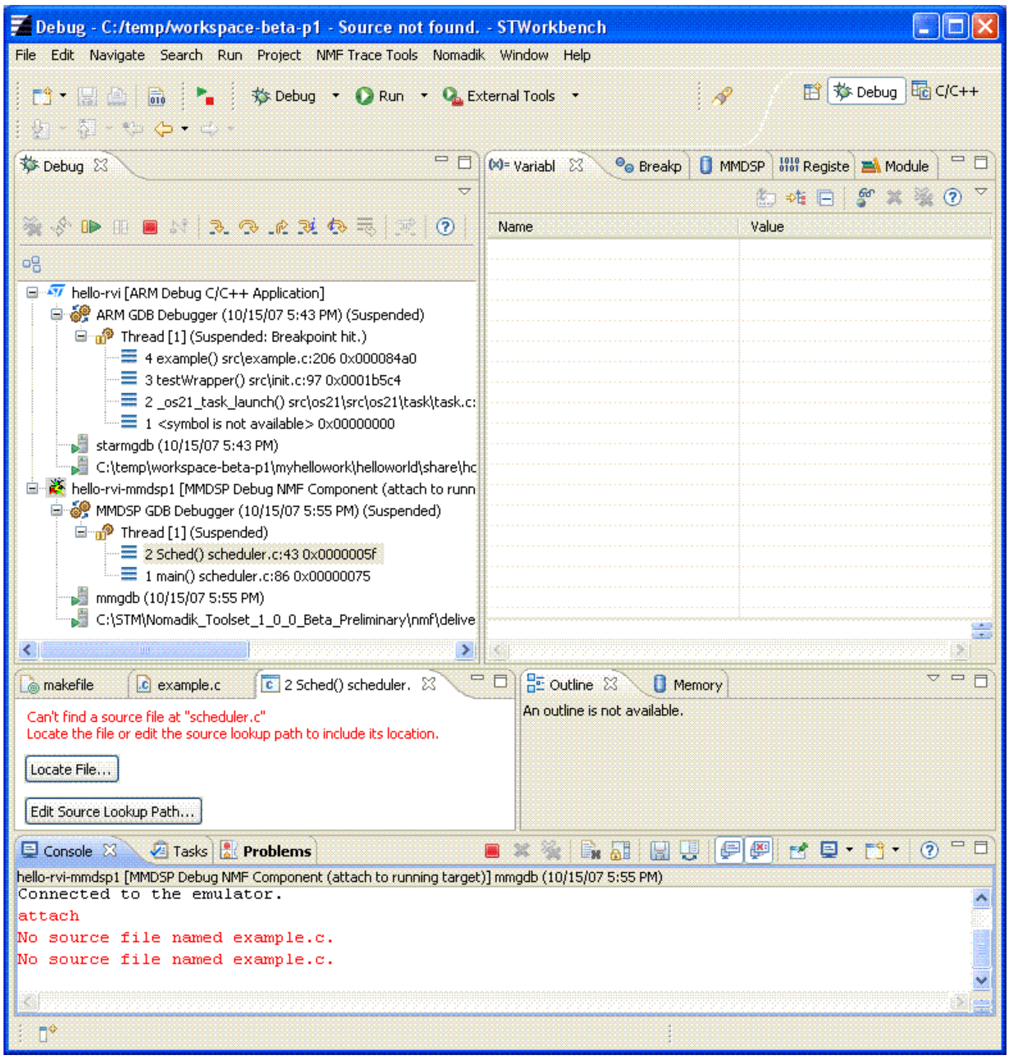
```
C:\temp\workspace-beta-p2\myhellowork\helloworld\share\host\src\example.c [line: 209]
C:\temp\workspace-beta-p2\myhellowork\helloworld\share\host\src\example.c [line: 221]
```

```
    /* Execute test, give datas to adder input
    /*****************************************
    for (loop_counter = 0; loop_counter < LOOP_
    {
        t_uint16 a = loop_counter;
        t_uint16 b = loop_counter + GAP_WITH_A_

        // Infinite loop until reaction is not
        while ((loop_counter - pCtx->reac_numb
        {
```

```
0x00000a08 <example+832>: mov     r6, r4, lsl #16
0x00000a0c <example+836>: mov     r6, r6, lsr #16
0x00000a10 <example+840>: mov     r7, r7, lsr #16
0x00000a14 <example+844>: b       0xbb8 <example+1264>
0x00000a18 <example+848>: streqh  sp, [r4], -r8
0x00000a1c <example+852>: svcvs   0x00727245
0x00000a20 <example+856>: strpl   r3, [r0, #-2674]!
0x00000a24 <example+860>: stfvse  f6, [r2], #-440
0x00000a28 <example+864>: svcvs   0x00742065
0x00000a2c <example+868>: stfvse  f6, [r12], #-128
```

3. Launch the MMDSP debug configuration. The ‘hot attach’ action stops the MMDSP core in the scheduler code, see *Figure 19*.

Figure 19. MMDSP core stopped in the scheduler code


Debug - C:/temp/workspace-beta-p1 - Source not found. - STWorkbench
File Edit Navigate Search Run Project NMF Trace Tools Nomadik Window Help
Debug
Run
External Tools
Debug
C/C++
Variabl
Breakp
MMDSP
Registe
Module
Name
Value
makefile
example.c
2 Sched() scheduler.
Locate File...
Edit Source Lookup Path...
Outline
Memory
Console
Tasks
Problems

```
hello-rvi [ARM Debug C/C++ Application]
  ARM GDB Debugger (10/15/07 5:43 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      4 example() src\example.c:206 0x000084a0
      3 testWrapper() src\init.c:97 0x0001b5c4
      2 _os21_task_launch() src\os21\src\os21\task\task.c:
      1 <symbol is not available> 0x00000000
  starmgdb (10/15/07 5:43 PM)
  C:\temp\workspace-beta-p1\myhellowork\helloworld\share\hc
hello-rvi-mmdsp1 [MMDSP Debug NMF Component (attach to runn
  MMDSP GDB Debugger (10/15/07 5:55 PM) (Suspended)
    Thread [1] (Suspended)
      2 Sched() scheduler.c:43 0x0000005f
      1 main() scheduler.c:86 0x00000075
  mmgdb (10/15/07 5:55 PM)
  C:\STM\Nomadik_Toolset_1_0_0_Beta_Preliminary\nmf\delive
```

Can't find a source file at "scheduler.c"
Locate the file or edit the source lookup path to include its location.

An outline is not available.

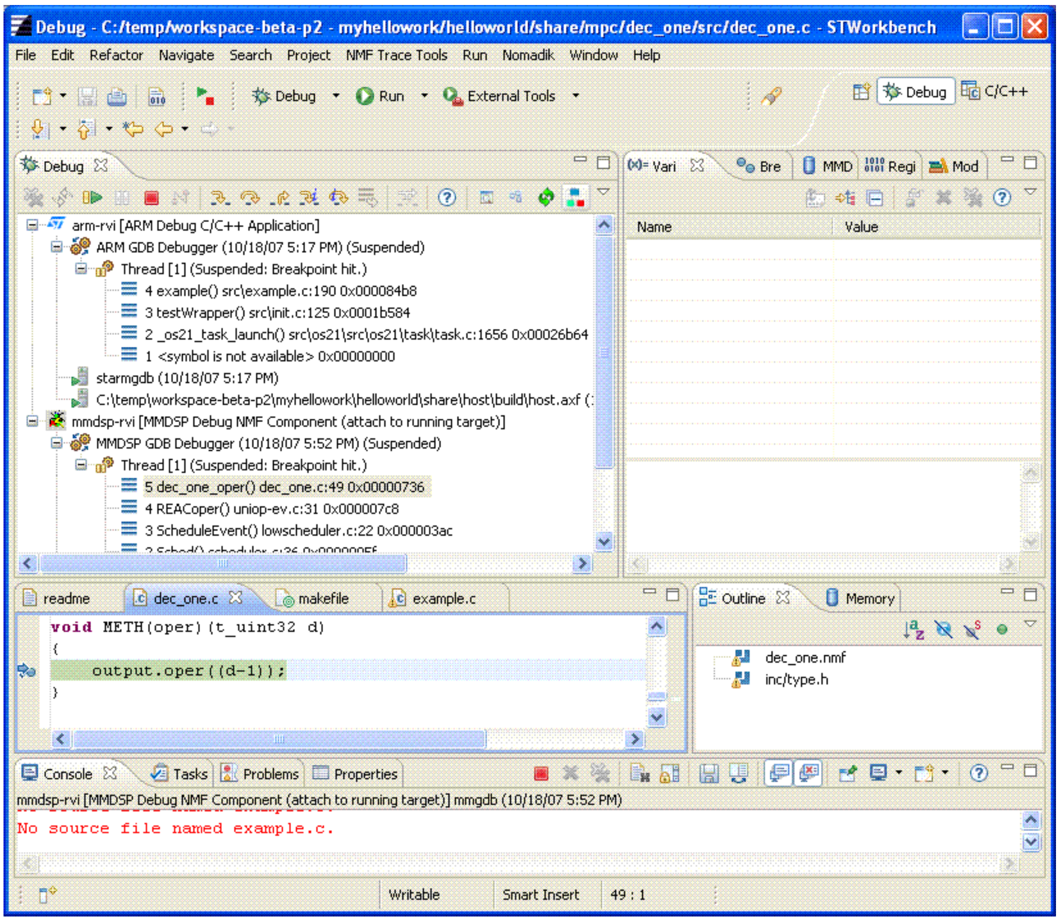
```
hello-rvi-mmdsp1 [MMDSP Debug NMF Component (attach to running target)] mmgdb (10/15/07 5:55 PM)
Connected to the emulator.
attach
No source file named example.c.
No source file named example.c.
```

4. Select the sources for the user code (for example `dec_one.c`), add a breakpoint in the `oper` method and continue the execution of the scheduler. The MMDSP thread should be in running mode.

5. Go to the ARM thread and continue the execution. The breakpoint in `oper` is hit in the MMDSP Core, see *Figure 20*.

Note: *To correctly restart a debug session on a NMF application, it requires a hardware reset of the Nomadik chip.*

Figure 20. Breakpoint hit in MMDSP component


Debug - C:/temp/workspace-beta-p2 - myhellowork/helloworld/share/mpc/dec_one/src/dec_one.c - STWorkbench
File Edit Refactor Navigate Search Project NMF Trace Tools Run Nomadik Window Help
Debug Run External Tools
Debug C/C++
Debug
Vari Bre MMD Regi Mod
Name Value
readme dec_one.c makefile example.c
Outline Memory
dec_one.nmf
inc/type.h
Console Tasks Problems Properties
Writable Smart Insert 49 : 1

```
arm-rvi [ARM Debug C/C++ Application]
  ARM GDB Debugger (10/18/07 5:17 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      4 example() src\example.c:190 0x000084b8
      3 testWrapper() src\init.c:125 0x0001b584
      2 _os21_task_launch() src\os21\src\os21\task\task.c:1656 0x00026b64
      1 <symbol is not available> 0x00000000
  starmgdb (10/18/07 5:17 PM)
  C:\temp\workspace-beta-p2\myhellowork\helloworld\share\host\build\host.axf (:
mmdsp-rvi [MMDSP Debug NMF Component (attach to running target)]
  MMDSP GDB Debugger (10/18/07 5:52 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      5 dec_one_oper() dec_one.c:49 0x00000736
      4 REACoper() uniop-ev.c:31 0x000007c8
      3 ScheduleEvent() lowscheduler.c:22 0x000003ac
```

```
void METH(oper)(t_uint32 d)
{
    output.oper((d-1));
}
```

```
mmdsp-rvi [MMDSP Debug NMF Component (attach to running target)] mmgdb (10/18/07 5:52 PM)
No source file named example.c.
```

4 Other types of projects

You can use the Nomadik Toolset to develop single-core software programs that run independently on any core of the Nomadik chip. These are described as MMDSP+ projects and ARM projects.

4.1 Nomadik MMDSP+ Projects

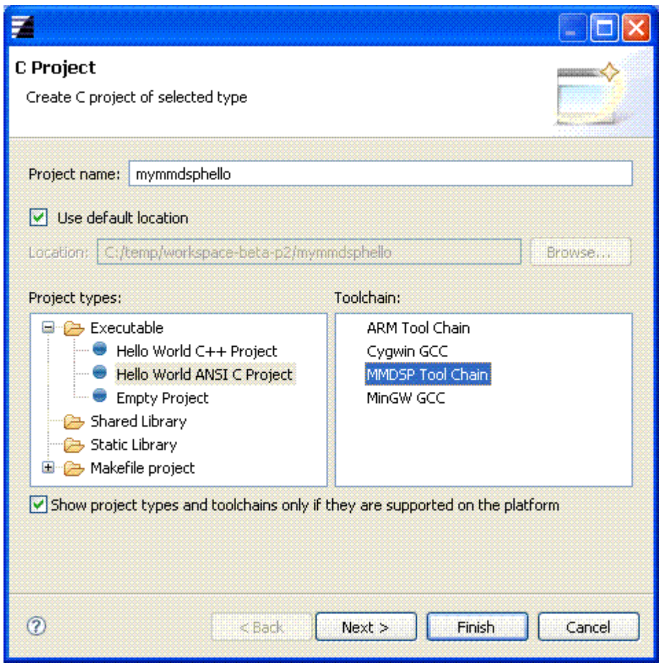
The Nomadik Toolset has the ability, inside the STWorkbench, to edit, compile and debug software programs that are intended to run in standalone mode on any MMDSP+ core of the Nomadik Chip.

If a component requires high CPU consumption, before this component is integrated into the test environment of a multimedia application, initial development can take place as a stand-alone MMDSP+ project. The advantage of this is that early evaluation and optimization of specific algorithms can be carried out before the component is integrated with the rest of the application. The MMDSP+ simulator provides specific profiler functionalities to assist in this task.

When testing on real Nomadik hardware, we recommend that the component is integrated into an NMF project. This is because test programs running under an NMF project are able to use specific test and measurement features that are not available to MMDSP+ projects.

MMDSP+ Projects can be managed either through `make` files and direct calls to the command line level MMDSP+ tools, or by creating an MMDSP+ project graphically managed under the STWorkbench IDE, see *Figure 21* for the creation of the "helloworld" example of an MMDSP executable type project.

Figure 21. Creating a "helloworld" MMDSP project



C Project

Create C project of selected type

Project name: mymmdsphello

☑ Use default location

Location: C:/temp/workspace-beta-p2/mymmdsphello

Project types:
- Executable
  - Hello World C++ Project
  - Hello World ANSI C Project
  - Empty Project
- Shared Library
- Static Library
- Makefile project

Toolchain:
- ARM Tool Chain
- Cygwin GCC
- MMDSP Tool Chain
- MinGW GCC

☑ Show project types and toolchains only if they are supported on the platform

Note: Only use MMDSP+ Projects when the overall software architecture definition has been finalized and the MMDSP+ main components are identified.

4.2 Nomadik ARM Projects

The Nomadik Toolset has the ability, inside the STWorkbench, to edit, compile and debug standalone ARM programs.

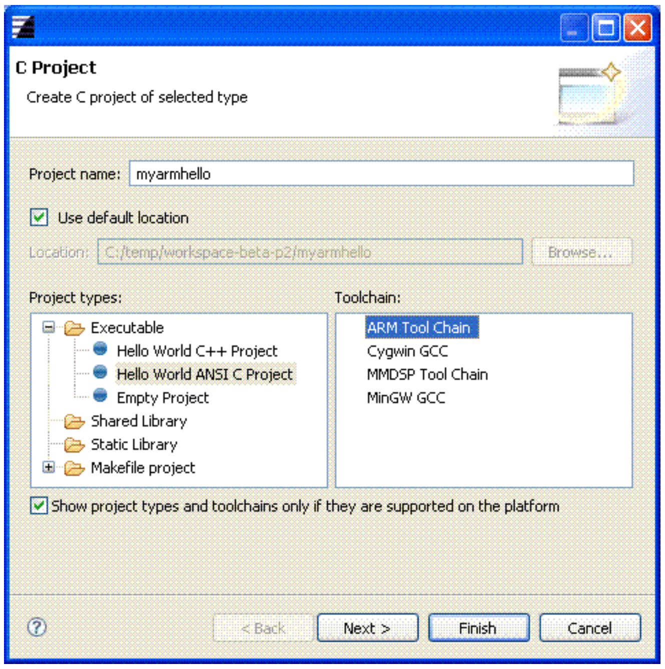
This can be useful for:

- hardware validation
- developing and maintaining the test harness for Multimedia applications
- developing parts of demo programs for the Multimedia applications

Nomadik ARM projects can benefit from the OS21 real-time Operating System environment and OSPlus services.

You can manage ARM projects either through make files and direct calls to the compiler command line, or create ARM projects graphically using the STWorkbench, see *Figure 22*.

Figure 22. Creating a “helloworld” ARM project



C Project

Create C project of selected type

Project name: myarmhello

[✓] Use default location

Location: C:/temp/workspace-beta-p2/myarmhello

Project types:
- Executable
  - Hello World C++ Project
  - Hello World ANSI C Project
  - Empty Project
- Shared Library
- Static Library
- Makefile project

Toolchain:
- ARM Tool Chain
- Cygwin GCC
- MMDSP Tool Chain
- MinGW GCC

[✓] Show project types and toolchains only if they are supported on the platform

5 Nomadik Toolset license

The Nomadik Toolset contains software provided under a variety of licenses. Some components are “free” or “open source” software, while other components are proprietary. This chapter explains how these licenses apply to the specific use of the Nomadik Toolset and further explains the user’s legal rights and obligations.

5.1 Overview

Table 1 lists the license terms applicable to each component of the Nomadik Toolset.

Table 1. Software licenses

Component	License
STMicroelectronics binary products: MMDSP Code generator & other Tools	STMicroelectronics Binary Code License Agreement.
The Java Runtime Environment.	SUN Code End-user License Agreement;
The ECLIPSE products.	Eclipse.org Software User Agreement. Eclipse Public License (EPL). They include software developed by the Apache Software Foundation and subject to the Apache License, Version 2.0
Portions of NOMADIK STWorkbench	STWorkbench Software License Agreement.
The GNU products.	GNU General Public License.
The newlib library.	Newlib License.
The libgloss library.	Libgloss License.
The GNU documentation.	GNU Free Documentation License
The BOOST Code.	BOOST Software License Agreement.
Graphwiz utilities	Common Public Licence

The description of each license, as well as a detailed and up to date description of the scope of each license inside the toolset, is present in the HTML pages located at the root of the toolset installation.

5.2 Consequence on Nomadik IP software

Although some of the licenses that apply to the Nomadik Toolset are “free software” or “open source software” licenses, none of these licenses impose any obligation on you to reveal the source code of applications you build with the Nomadik Toolset. You may develop proprietary applications and libraries with the Nomadik Toolset.

6 Revision history

Table 2. Document revision history

Date	Revision	Changes
24-Jan-2008	A	Initial release.