ST20 Embedded Toolset R2.3 Delivery Manual

Toolset version R2.3.1

ADCS 7257995 Rev. K

Contents

Preface

ST20 documentation suite

The document set provided with the toolset comprises the documents described below.

The documentation is automatically installed with the toolset, in PDF, HTML and Microsoft Compiled Help (Windows only) formats. PDF files can be read using Adobe Acrobat Reader, which can be obtained from the Adobe website at www.adobe.com. In addition, various miscellaneous HTML and text files are installed, including readme files for the code examples.

ST20 Embedded Toolset Delivery Manual (this document)

The delivery manual provides installation instructions, a summary of the release, a list of changes since the previous revision, and any other material that will be included in future releases of the other manuals.

ST20 Embedded Toolset User Manual (ADCS 7143840)

This manual provides an overview of the toolset and a getting started guide for using the graphical user interface. It also describes how the core features of the toolset are used to build and run application programs. It includes compiling and linking, connecting to a target, loading programs and application debugging.

The ST Visual Develop GUI is not supported on UNIX and Linux platforms. UNIX and Linux users should continue to use the UNIX **st20run** GUI (the GUI used in all releases prior to R2.0), which is documented as an appendix in the User Manual.

OS20 User Manual (ADCS 7473749)

This manual is a user guide for the OS20 real-time kernel. It provides an introduction and getting started section then continues with separate chapters for each of the main features supported, such as, kernel, memory and partitions management, tasks, semaphores, message handling queues, real-time clocks and interrupts.

ST20 Embedded Toolset Reference Manual (ADCS 7250966)

This manual describes the advanced facilities of the toolset such as the assembler, librarian, lister, and ST20 instruction set simulator. It also describes facilities such as code and data

placement, the stack depth and memory map files, the use of relocatable code units and profiling and trace facilities. Reference information is provided for the C and C++ implementations, the libraries and the command language.

Conventions used in this manual

Typographical conventions

The following typographical conventions are used in this manual:

Bold	Used within text for emphasis
Blue Italic	Denotes hyperlink cross-references
Italic	Denotes variables or nonhyperlink cross-references
UPPER CASE	Denotes special terminology, for example, register or signal/pin names
`Monotype bold`	Denotes command options, command line examples, code fragments, and program listings
`Monotype bold italic`	Denotes arguments or parameters in command syntax definitions
`Monotype italic`	Denotes code comments
Braces {}	Denotes items which may be repeated in command syntax
Brackets []	Denotes optional items in command syntax
Ellipsis **`...`**	In general terms, used to denote the continuation of a series. For example, in syntax definitions denotes a list of one or more items
|	In command syntax, separates two mutually exclusive alternatives
`__`	Denotes two underscores together. Any space visible between the two underscores should be ignored

Command line conventions

Example command lines and directory path names are documented using UNIX style notation which makes use of the forward slash (`/`) delimiter. In most cases this should be recognized by Windows hosts; if not, substitute the forward slash with a backslash (`\`). For example, the directory:

```
release/examples/simple
```

is the same as:

```
release\examples\simple
```

Command line options are prefixed by a hyphen (`-`); this is compatible with Windows and UNIX.

Examples of the debugging tools use the following convention to distinguish command prompts:

"`%`" is used to indicate the operating system command prompt, for example:

```
% st20run ...
```

"`>`" is used to indicate the interactive command language prompt, for example:

```
> break ...
```

Acknowledgements

Linux® is a registered trademark of Linus Torvalds.

Red Hat® is a registered trademark of Red Hat Software, Inc.

Sun and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the US and other countries.

UNIX® is a registered trademark of the The Open Group in the US and other countries.

Microsoft®, Windows® and Visual Studio®are registered trademarks of Microsoft Corporation in the United States and other countries.

The Warp Nine driver par1284.sys is copyright Warp Nine Engineering San Diego,California.

The C compiler implementation was developed from the Perihelion Software “C” Compiler and the Codemist Norcroft “C” Compiler.

The C++ language front-end was developed under a Licence Agreement between Edison Design Group, Inc. (EDG) and STMicroelectronics Limited.

The C++ library is supplied by and is copyright of Dinkumware Ltd.

This product incorporates innovative techniques which were developed with support from the European Commission under the ESPRIT Projects:

- P2701 PUMA (Parallel Universal Message-passing Architectures),
- P5404 GPMIMD (General Purpose Multiple Instruction Multiple Data Machines),
- P7250 TMP (Transputer Macrocell Project),
- P7267 OMI/STANDARDS,
- P6290 HAMLET (High Performance Computing for Industrial Applications),
- P606 STARLIGHT (Starlight core for Hard Disk Drive Applications).

1 Introduction

This document accompanies the R2.3 product release (R2.3.1) of the ST20 Embedded Toolset. This product supersedes all previous ST20 Embedded Toolset releases, and addresses all major problems identified.

Users must register with customer support that they have installed the R2.3 toolset (see the *Installation* chapters).

The package includes a toolchain, debugger and run-time kernel for the ST20-C1 and ST20-C2 cores, targeting all existing ST20 silicon parts containing a ST20-C1 or ST20-C2 core and a DCU (Diagnostic Control Unit). ST20-C1 and ST20-C2 simulators are included, which can be used as targets.

Supported hosts are listed in *Appendix A: Referenced technology on page 31*.

All versions of the toolset provide ANSI C and C++ toolchains.

Library files (excluding toolset libraries and files containing C++) produced with the toolset will be supported by future versions of the toolset. Linker output files may not be supported by future versions of the toolset.

This release incorporates all the patch releases of the previous toolset release (if any) and includes an update of the documentation set. It addresses the high-priority bug fixes and enhancements.

Customers using the chip STi5100 must refer to *Appendix D.6: STi5100 cut3 and earlier on page 62*.

Customers using the ST20-C10x core (for example, chips STi7710 and STi5105) must refer to *Appendix D.7: ST20-C105 and ST20-C106 cores on page 62*.

1.1 Structure of this manual

Chapter 2: New features gives a summary of the new features in this version of the toolset. All new functionality is fully described in the appropriate manuals. Changes since all previous versions of the toolset are listed in *Appendix B: Toolset revisions on page 33*. *Appendix C: Porting code from earlier toolsets on page 53* gives detailed instructions for porting code from earlier toolsets.

Chapter 3: Contents of the release gives the contents of the toolset. *Chapter 7: Release directories on page 23* gives the directory structure of the installed toolset.

Chapter 4: Installing the PC Windows release, *Chapter 5: Installing the Linux release* and *Chapter 6: Installing the Solaris release* give instructions for installing the toolset.

Chapter 8: Release notes includes notes that apply only to this release, some late changes in functionality that were not included in the other manuals and a short documentation errata.

Appendix D: Silicon bugs that affect the toolset on page 59 is a list of known silicon bugs that affect software development.

1.2 Compatibility with previous versions

This release is backwards compatible with all releases from R1.7 onwards; no changes are required to source code, except:

- From R1.8.1, some library functions have been moved to different library files.
- From R1.8.1, the default effect of the order of **st20cc** parameters has changed. A new option is provided to keep the effect the same as in R1.8 and earlier toolsets.
- The R1.8.1 and later **`endorder`** behavior is not compatible with previous releases. If **`endorder`** is used without a priority in a command language procedure called using the **st20cc** option **`-p`**, the functions now run after the program exit.
- From R1.8.1, OS20 interrupt handler library changes. See the *Interrupts* chapter of the *OS20 User Manual*.
- The implementation of C++ in this release is not compatible with C++ implementation in earlier toolset releases. Existing C++ modules will need to be recompiled.

Compatibility issues with porting from toolsets before R1.7 are described in *Appendix C: Porting code from earlier toolsets on page 53*.

1.3 ST Visual Develop (st20dev)

The Windows IDE provided in this release and is known as ST Visual Develop (**st20dev**). This is not available for Linux and Solaris hosts.

Linux and Solaris users should use the graphical interface to **st20run**. This is not available for Windows.

Both graphical interfaces are documented in separate appendices in the *ST20 Embedded Toolset User Manual*.

1.4 Comprehensive C++ support

The toolset supports a subset of the ISO/IEC 14882 C++ standard, including templates and exceptions. The C++ Standard Library (which includes the Standard Template Library) is included.

Important C++ users **must** recompile all sources when migrating from earlier toolsets.Compatibility of C++ objects, libraries, linked unit (`.lku`) and debug (`.dbg`) files built with R2.3.1 cannot be guaranteed with future toolsets; C++ users may have to recompile all sources when migrating to future toolsets.

1.5 Support

For product support, contact your local ST Field Applications Engineer (FAE).

2 New features

The new features in this release since the previous release include:

- Supported on Red Hat Linux Enterprise Workstation Version 3 or compatible version
  - ➢ Earlier versions no longer supported
- OS20 updated to version 2.11.06:
  - ➢ time compare allows for equals as well as greater than a value (INSbl26941)
  - ➢ fixed failure to pre-empt a low priority task when performing scheduling that interrupts **`task_unlock`** (INSbl27338)
  - ➢ **`timer_init_pwm`** uses software to divide PWM4 ticks by 256 (INSbl26999)
- compiler support to align string literals to short or word boundary for wide char handling
- Support for a new host target interface ST Micro Connect 2
  - ➢ The term *ST Micro Connect* is now used as a generic description for ST Micro Connect 1 or ST Micro Connect 2
  - ➢ Previously, ST Micro Connect 1 was just called ST Micro Connect

There are also numerous minor bug fixes, updates and enhancements.

2.1 Bug fixes

Many known defects from previous toolset versions have been fixed. These are listed, with enhancements and outstanding defects, in the bug list. The bug list is in HTML format and may be found in the installed toolset in the **`doc`** directory and on the toolset CD-ROM.

2.2 Support for new silicon parts

There are no new ST chips supported since the previous release.

3 Contents of the release

The ST20 Embedded Toolset release contains the following components:

- compiler/linker driver (**st20cc**),
- C++ translation tools called by **st20cc**,
- librarian tool (**st20libr**),
- file lister tool (**st20list**),
- program load run and debug tool (**st20run**),
- full ANSI C library and header files,
- ST20-specific libraries for maths and debugging with header files,
- C++ support libraries and header files,
- OS20 real-time kernel library and header files,
- instruction set simulator (**st20sim**),
- configuration files,
- a set of example files, including a set of example programs using OS20,
- sources of the OS20 real-time kernel library,
- sources of the system start-up code and the breakpoint traphandler,
- ST20 Embedded Toolset bug list.

The PC Windows release includes the *ST Visual Develop* (**st20dev**) integrated development environment.

The UNIX and Linux releases include the **st20run** GUI.

For a description of the differences between this and previous releases, see *Chapter 2: New features on page 5* and *Appendix B: Toolset revisions on page 33*.

4 Installing the PC Windows release

This chapter describes how to install the PC Windows release. For instructions on installing the toolset on other platforms, see *Chapter 5: Installing the Linux release on page 13* and *Chapter 6: Installing the Solaris release on page 19*.

The PC Windows release is available on CD-ROM and in compressed files available on the STMicroelectronics FTP server or website.

4.1 Prerequisites

The following are required in order to use this release:

- an Intel-compatible PC running Windows 2000 or Windows XP (see *Appendix A: Referenced technology on page 31* for further details),
- a CD-ROM drive, or access to the STMicroelectronics FTP site or website,
- approximately 75 Mbytes of free disk space.

Note: In addition, an ST Micro Connect or a JEI is required for communicating with silicon targets.

4.2 Licensing and registering

Before installing the toolset, the installer and users must read and agree the licence conditions given in the installation.

Installers should also register their installation in order to receive support. The installation program tries to automatically register the installation by sending an e-mail to **st20.register@st.com**. The e-mail contains the following information:

- name of installer,
- company,
- location,
- country,
- product (for example, ST20 Embedded Toolset),
- toolset version (for example, R2.3.1),
- host platform (for example, Windows XP).

If e-mail is not available on the installation machine, or the e-mail is not sent automatically for any other reason, please send an e-mail containing the above information to **st20.register@st.com**.

4.3 Installation

This release includes the toolset and the documentation in PDF, HTML and Microsoft Compiled Help formats.

Note: When installing for Windows platforms ensure you have administator privilege, this is required in order to update system DLLs.

4.3.1 Installation directory

The default installation directory is **STM\ST20R2.3.1** on the system drive. A different directory may be selected by entering the path name of the directory in the appropriate installation program dialog box.

Note: The toolset does not support spaces in directory or file names so avoid using directory names such as **My Documents**.

4.3.2 Installing from CD-ROM

Run the installation program **stm-st20.231-2.3.1-MSWin32-x86.exe** (which can be found in the **pc** directory of the CD-ROM) and follow the instructions from there.

This copies the release into the installation directory, installs the documentation and sets up the environment.

4.3.3 Installing from FTP

On the STMicroelectronics FTP server there is a compressed file available for the Windows installation, **stm-st20.231-2.3.1-MSWin32-x86.exe**. This file is used to install the toolset into the installation directory, install the documentation and set up the environment.

1. Download the self-extracting file **stm-st20.231-2.3.1-MSWin32-x86.exe** from the STMicroelectronics FTP server.
2. Transfer the compressed file into a temporary directory.
3. Run the installation program **stm-st20.231-2.3.1-MSWin32-x86.exe** in the temporary directory, and follow the instructions from there. This copies the release to the installation directory, sets up the environment and installs the parallel port device driver.

4.4 Setting up the environment

If the installation has worked correctly, no further setting up is required. The following sections contain notes on what is done by the installation program to set up the environment and the parallel port to aid in solving problems with the set up process.

As part of the installation process a batch file called **st20.bat** is created at the top-level of the release directory which sets up the environment to run the tools from a command prompt. The installation process also offers the option to update the environment in the system properties.

Under Windows, if the user installing the toolset has system administrator privileges then the environment for all users (that is, the system environment) will be updated, instead of just the environment for the user installing the toolset.

A menu called **STM Tools** is also added to the **Start** button on the task bar that contains a number of shortcuts to various tools in the toolset, and a shortcut to a command prompt which is set up to run the **st20.bat** batch file automatically when it is opened.

The following sections describe the actions of the **st20.bat** batch file.

4.4.1 Setting up your path

To be able to use the tools, the **bin** subdirectory of the installation directory must be on your path. For example, for the default installation directory on the C: drive:

```
set PATH=C:\STM\ST20R2.3.1\bin;%PATH%
```

4.4.2 Setting environment variables

To enable the tools to find libraries and include files you must set up the environment variable **ST20ROOT**. This environment variable should contain the path of the release directory, under which the standard library and include file directories will be found. For example, for the default installation directory on the C: drive:

```
set ST20ROOT=C:\STM\ST20R2.3.1
```

If ST Visual Develop is to be used exclusively to drive **st20cc** and **st20run**, it is not necessary to set the environment variable **ST20ROOT**; see the chapter *Introduction* in the *ST20 Embedded Toolset User Manual*.

4.4.3 The st20.rc file

The file **st20.rc** in the toolset installation directory contains some predefined commands for the toolset; it is read by some of the tools on start up.

4.5 Setting up the target interface

The Windows release supports the following interfaces to the target system:

- ST Micro Connect using Ethernet, parallel port or USB interface,
- JEI Ethernet interface.

4.5.1 ST Micro Connect

A CD-ROM is supplied with the ST Micro Connect, which includes all necessary device drivers and any necessary Windows upgrades. Once this software has been installed on your PC, no further action is needed to set up the target interface.

4.6 Checking the installation

The installation can be checked by running the getting started example program. The program may be run on the simulator, so no target hardware is needed.

1 Change directories to the getting started example directory. For example, for the default installation directory:

```
cd /STM/ST20R2.3.1/examples/getstart
```

This contains several files including **hello.c** and **sti5500.cfg**.

Note: Make a copy of the examples directory and use the copy to experiment with, this will keep the originals safe.

2 Compile and link:

```
st20cc hello.c -T sti5500.cfg -p link -g
```

The output file should be called **hello.lku**.

3 To run the program on the simulator, enter the command:

```
st20run -i sti5500.cfg -t eval5500sim hello.lku
```

The program displays the message:

```
Hello World from Osprey
```

4 To start ST Visual Develop in debug mode, use the command line:

```
st20dev -i sti5500.cfg -t eval5500sim hello.lku
```

Select **Start Debug** > **Go** from the **Build** menu (or press **F5**) to run the program. To close ST Visual Develop, select **Exit** from the **File** menu.

Installing the Linux release

5

This chapter describes how to install the Linux release. For instructions on installing the toolset on other platforms, see *Chapter 4: Installing the PC Windows release on page 9* and *Chapter 6: Installing the Solaris release on page 19*.

The Linux release is available on CD-ROM and in compressed files available on the STMicroelectronics FTP server or website.

5.1 Prerequisites

The following are required in order to use this release:

- an Intel-compatible PC running Linux (for version information see *Appendix A: Referenced technology on page 31*),
- a CD-ROM drive, or access to the STMicroelectronics FTP site or website,
- approximately 65 Mbytes of free disk space.

Note: In addition, an ST Micro Connect or a JEI is required for communicating with silicon targets.

5.2 Licensing and registering

Before installing the toolset, the installer and users must read and agree the licence conditions given in the text document in the root directory of the toolset CD-ROM or provided with the `tar` file on the FTP site.

Installers should also register their installation in order to receive support. Registration is carried out by sending an e-mail to `st20.register@st.com` with the following information:

- name of installer,
- company,
- location,

- country,
- toolset version (for example, R2.3.1),
- product (that is, ST20 Embedded Toolset),
- host platform (that is, Linux).

5.3 Installation

Root privilege is normally needed to install the toolset, and the Red Hat installation tool RPM is normally used. However, if either root privilege or RPM is not available the toolset can still be installed, as described in *Section 5.3.4: Installing without RPM or without root privilege*.

5.3.1 Installation directory

Choose a location in your filing system and a name for the installation directory. The default installation directory is **`/opt/STM/ST20R2.3.1`**. The RPM is relocatable by replacing **`/opt`** with a different directory root, so a different installation directory can be selected; see *Section 5.3.5* and the RPM documentation for details.

For the rest of this section, the installation directory will be referred to as **`$RELEASE`**. This notation may be used in the following commands by setting an environment variable to the path name of the directory. For example, using the default **`/opt/STM/ST20R2.3.1`** as the installation directory, then using the C shell (csh):

```
setenv RELEASE /opt/STM/ST20R2.3.1
```

Using the Bourne shell (sh):

```
RELEASE=/opt/STM/ST20R2.3.1
export RELEASE
```

Note: **`$RELEASE`** *does not control the location of the installation.*

5.3.2 Installing from CD-ROM

1 As root user, mount the CD-ROM, for example:

```
mount /mnt/cdrom
```

The Linux toolset is in the file named **`stm-st20.231-2.3.1-1.i386.rpm`** in the **`linux`** directory.

2 Use the RPM installation utility to install the toolset, for example:

```
rpm -ivh /mnt/cdrom/linux/stm-st20.231-2.3.1-1.i386.rpm
```

This copies the release into the installation directory, installs the documentation set and updates the system file **`/etc/ld.so.conf`**.

5.3.3 Installing from FTP

On the STMicroelectronics FTP server there is a compressed file available for the PC Linux installation, **stm-st20.231-2.3.1-1.i386.rpm**. This file is used to install the toolset into the installation directory and set up the environment.

1 Download this file into the **$RELEASE** directory.

2 As the root user, use the RPM installation utility to install the release, for example:

```
rpm -ivh stm-st20.231-2.3.1-1.i386.rpm
```

This copies the release into the installation directory, installs the documentation and updates the system file **/etc/ld.so.conf**.

The file **stm-st20.231-2.3.1-1.i386.rpm** can now be deleted.

5.3.4 Installing without RPM or without root privilege

These instructions should only be followed if for some reason it is impossible to use RPM, or root privilege is not available.

1 Either download the RPM file from the FTP server (see *Section 5.3.3*) or mount the CD-ROM (see *Section 5.3.2*).

2 Choose an installation directory, and set the variable **RELEASE** to the path name of that directory. For example, using **~/STM/ST20R2.3.1**, in a C shell:

```
setenv RELEASE ~/STM/ST20R2.3.1
```

The equivalent in a Bourne shell:

```
RELEASE=~/STM/ST20R2.3.1
export RELEASE
```

3 Create the installation directory with **mkdir(1)**:

```
mkdir -p $RELEASE
```

4 In the directory where the RPM files have been downloaded, unpack the distribution and copy it to the **$RELEASE** directory, as follows:

```
rpm2cpio stm-st20.231-2.3.1-1.i386.rpm | cpio -vid
mv opt/STM/ST20R2.3.1/* $RELEASE
rmdir -p opt/STM/ST20R2.3.1
```

5 Add the library directory **$RELEASE/lib** to **LD_LIBRARY_PATH**.

For example, using a C shell, if **$LD_LIBRARY_PATH** already exists:

```
setenv LD_LIBRARY_PATH $RELEASE/lib:$LD_LIBRARY_PATH
```

If **$LD_LIBRARY_PATH** does not already exist:

```
setenv LD_LIBRARY_PATH $RELEASE/lib
```

Using a Bourne shell this becomes:

```
LD_LIBRARY_PATH=$RELEASE/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

or:

```
LD_LIBRARY_PATH=$RELEASE/lib
export LD_LIBRARY_PATH
```

The file **stm-st20.231-2.3.1-1.i386.rpm** can now be deleted.

5.3.5 Changing the location of the default installation

If, as root user, it is not desired that the root directory of the release is **/opt**, then the package can be relocated using the **rpm --relocate** option. For example, to install in the directory **/usr**:

```
rpm -ivh stm-st20.231-2.3.1-1.i386.rpm --relocate '/opt=/usr'
```

This will copy the release into the directory **/usr/STM/ST20R2.3.1**. See the RPM documentation for further details.

5.4 Setting up the environment

Having installed the release there are several environment variables to be set up before you can use any of the tools. These variables are described in this section.

5.4.1 Setting up your paths

To be able to use the tools you will need to add the directory **$RELEASE/bin** to your **PATH**, and **$RELEASE/lib** to your **LD_LIBRARY_PATH**.

For example, using a C shell:

```
setenv PATH $RELEASE/bin:$PATH
setenv LD_LIBRARY_PATH $RELEASE/lib:$LD_LIBRARY_PATH
```

Using a Bourne Shell, this becomes:

```
PATH=$RELEASE/bin:$PATH
export PATH

LD_LIBRARY_PATH=$RELEASE/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

If the RPM was installed as root then setting **$LD_LIBRARY_PATH** is not required as the system file **/etc/ld.so.conf** will already have been updated to contain the suitable value, see *Section 4.3.3: Installing from FTP on page 10*.

5.4.2 Setting environment variables

To enable the tools to find libraries and include files you must set up the environment variable **ST20ROOT**. This environment variable should contain the path of the release directory, under which the standard library and include file directories will be found.

For example, using a C shell:

```
setenv ST20ROOT $RELEASE
```

Using a Bourne Shell, this becomes:

```
ST20ROOT=$RELEASE
export ST20ROOT
```

The environment variable **$HOME** should be set to point to the user's home directory.

5.4.3 The .st20rc file

A file called **.st20rc** has been installed into the **$RELEASE** directory. This contains some predefined commands for the toolset, and is read by some of the tools on start up.

5.4.4 Setting up the debugger GUI resource file

As part of the installation, an X resource file named **st20run** is available in the directory **$RELEASE/lib**. This file needs to be copied to the system wide **app-defaults** directory. The standard location for this directory is:

```
/usr/openwin/lib/app-defaults
```

However, this directory can be changed by setting the **XFILESEARCHPATH** environment variable (see man page **X11(7)**: **man -s 7 X11**).

If you wish to override the standard fonts then you should copy the **st20run** X resource file to your home directory (specified in **$HOME**) or to a directory referenced by the environment variable **XAPPLRESDIR**, and then amend this copy as desired.

5.5 Checking the installation

The installation can be checked by running the getting started example program. The program may be run on the simulator, so no target hardware is needed.

1 Change directories to the getting started example directory. For example, for the default installation directory:

```
cd $ST20ROOT/examples/getstart
```

This contains several files including **hello.c** and **sti5500.cfg**.

Note: You may wish to make a copy of the examples directory so that you can change and experiment with the examples, keeping the originals safe.

2 Compile and link:

```
st20cc hello.c -T sti5500.cfg -p link -g
```

The output file created is called **hello.lku**.

3 To run the program on the simulator, enter the command:

```
st20run -i sti5500.cfg -t eval5500sim hello.lku
```

The program displays the message:

```
Hello World from Osprey
```

4 To start the debugger graphical interface, use the command line:

```
st20run -i sti5500.cfg -t eval5500sim hello.lku -g
```

This opens the code window. Click on the **Go** button to run the program. To close down the debugger, click **File** > **Exit** > **No save**.

6 Installing the Solaris release

This chapter describes how to install the Solaris release. For instructions on installing the toolset on other platforms, see *Chapter 4: Installing the PC Windows release on page 9* and *Chapter 5: Installing the Linux release on page 13*.

The Solaris release is available on CD-ROM and in compressed files available on the STMicroelectronics FTP server or website.

6.1 Prerequisites

The following are required in order to use this release:

- a Sun 4 running Solaris (for version information, see *Appendix A: Referenced technology on page 31*),
- a CD-ROM drive, or access to the STMicroelectronics FTP site or website,
- approximately 70 Mbytes of free disk space.

Note: In addition, an ST Micro Connect or a JEI is required for communicating with silicon targets.

6.2 Licensing and registering

Before installing the toolset, the installer and users must read and agree the licence conditions given in the text document in the root directory of the toolset CD-ROM or provided with the **tar** file on the FTP site.

Installers should also register their installation in order to receive support. Registration is carried out by sending an e-mail to **st20.register@st.com** with the following information:

- name of installer,
- company,
- location,
- country,
- toolset version (for example, R2.3.1),
- product (that is, ST20 Embedded Toolset),
- host platform (that is, Solaris).

6.3 Installation

If you have a previous release installed then this release should be installed in a different directory. Do not install this release over an earlier release.

gunzip is available if required on the FTP site to decompress the transferred files.

The toolset includes documentation in PDF and HTML formats, as described in the *Preface on page ix*.

6.3.1 Creating a directory for the release

Choose a location in your filing system and a name for the installation directory. For the rest of this section, the installation directory is referred to as **$RELEASE**. This notation may be used in the following commands by setting an environment variable to the path name of the directory. For example, using **/opt/STM/ST20R2.3.1** as the installation directory:

```
setenv RELEASE /opt/STM/ST20R2.3.1
```

Create the installation directory with **mkdir(1)**:

```
mkdir $RELEASE
```

The equivalent in Bourne shell (sh):

```
RELEASE=/opt/ST/ST20R2.3.1
export RELEASE
```

6.3.2 Installing from CD

Extract the file **stm-st20.231-2.3.1-sun4-solaris.tar.gz**, in the **sun** directory, to the installation directory as follows:

```
cd $RELEASE
```

Use **gunzip** to decompress the **stm-st20.231-2.3.1-sun4-solaris.tar.gz** file, and then extract the release from the **tar** file, as follows:

```
gunzip /cdrom/cdrom0/sun/stm-st20.231-2.3.1-sun4-solaris.tar.gz
tar xvf /cdrom/cdrom0/sun/stm-st20.231-2.3.1-sun4-solaris.tar
```

6.3.3 Installing from FTP

On the STMicroelectronics FTP server, the file is available to install the toolset into the installation directory.

1. Download the file **stm-st20.231-2.3.1-sun4-solaris.tar.gz** from the STMicroelectronics FTP server.
2. Transfer the file into the **$RELEASE** directory.

Note: The FTP site also contains a copy of **gunzip** *(the GNU file decompression utility), which can be transferred into a utility directory.*

3. Use **gunzip** to decompress the **stm-st20.231-2.3.1-sun4-solaris.tar.gz** file, and then extract the release from the **tar** file, as follows:

```
gunzip stm-st20.231-2.3.1-sun4-solaris.tar.gz
tar xvf stm-st20.231-2.3.1-sun4-solaris.tar
```

6.4 Setting up the environment

Having installed the release there are several environment variables to be set up before you can use any of the tools. Setting up the environment for Solaris is exactly the same as setting up the environment for Linux; see *Section 5.4: Setting up the environment on page 16* for details.

6.5 Checking the installation

The installation may be checked by running the getting started example program. The program may be run on the simulator, so no target hardware is needed.

The procedure for checking the installation is identical to that for Linux users and is explained in *Section 5.5: Checking the installation on page 17*.

7 Release directories

The installation procedure creates the directory structure shown in *Table 1*. Some of these directories are described in more detail below.

Directory	Contents
`bin`	The tools.
`board`	Example configuration files for target boards.
`doc`	The documentation set.
`examples`	Example programs and configuration files.
`include`	Library header files.
`include/chip`	Memory address header files.
`lib`	Library files.
`src`	Source code of OS20, system start-up code and the debugger trap handler.
`stdcfg`	Standard configuration files.

Table 1: Release directories

As well as including these directories, the home directory also includes a file, **`index.htm`**. This file can be used to navigate the information supplied with the installation.

7.1 The board directory

The **`board`** directory contains configuration files for a selection of target evaluation boards supplied by STMicroelectronics which the ST20 R2.3 toolset has been validated with.

These configuration files may be used to run simple examples on the supported boards and they may be used as examples in writing configuration files for other boards. A **`readme.txt`** text file is provided which explains the configuration files.

Note: These are examples, not definitive configurations. Contact customer support or your FAE to obtain suitable configurations for your particular application.

7.2 The documents directory

The **doc** directory contains the supporting documentation supplied with the toolset. Several HTML files are provided to navigate the documentation; these can all be accessed from the **index.htm** file in the toolset CD-ROM root directory, and are explained in *Table 2*.

File	Description
acknow.htm	The acknowledgements page.
acroread.htm	Instructions on installing and using Acrobat Reader.
buglist.htm	Information on bugs outstanding and resolved in this release.
cdmap.htm	A map of the information provided.
docbug.htm	Instructions on how to get support on the toolset and report problems in the documentation.
docs.htm	This file displays a list of the documentation supplied with the toolset. Each document can be accessed from this page by downloading the relevant file.
licence.htm	The ST20 micro toolset software licence. This licence must be read and agreed to prior to use of the software.

Table 2: Contents of the doc directory

There are two subdirectories provided in the **doc** directory. These are explained in *Table 3*.

Directory	Description
images	The images used in the documentation.
st20	The documentation files. These can be accessed from the **docs.htm** file.

Table 3: The doc subdirectories

7.3 The examples directory

The examples are held in the **examples** directory. Each example has a **readme.txt** text file describing the example and makefiles to build the example.

Getting started examples

The **getstart** subdirectory contains a number of example files; see the *Quick start guide* chapter of the *ST20 Embedded Toolset User Manual* for how to compile and run these.

The **st20dev** subdirectory contains example files used for demonstrating ST Visual Develop:

- **build_and_debug** - see the *Building with ST Visual Develop* and *Debugging with ST Visual Develop* chapters in the *ST20 Embedded Toolset User Manual* for details.
- **tracing** - see the *Tracing and profiling* chapter in the *ST20 Embedded Toolset User Manual* for details.
- **projects_and_makefiles**.

OS20 examples

The **os20** subdirectory contains some examples of programs using OS20 features.

- The **getstart** subdirectory contains a terminal emulator, as an example of using OS20.
- The **c1int** and **c2int** subdirectories contain examples using the interrupt functions in the ST20-C1 and ST20-C2 implementations of OS20.
- The **c1timer** subdirectory contains an example of how to set up the timer device drivers for a ST20-C1 core so that OS20 can provide time functions.
- The subdirectory **task** contains some basic multi-tasking program examples using features of OS20.
- The **mutex** subdirectory contains a mutual exclusion example.
- The **dl** subdirectory contains example code for dynamic code loading under OS20.
- The **scripts** subdirectory contains example scripts for dumping OS20 data structures.

Other examples

- The **dl** subdirectory contains an example showing how to call functions in dynamically loaded relocatable code units (RCUs). The example illustrates the use of the **export** and **import** command language functions.
- The **dualboot** subdirectory contains an example of booting from a bootstrap ROM into an application ROM. This example uses the facilities provided in toolset versions R1.8.1 or later, but an example is also provided in **dualboot-pre1.8.1** of how to perform the same function with older toolsets.
- The **flash** subdirectory contains an example FLASH ROM programming application.
- The **memio**, **minifs** and **uartio** subdirectories contain examples of using the POSIX I/O API to read and write to a device.
- The **prepoke** subdirectory contains an example showing how to use the **PreLoopCallback** and **PostLoopCallback** functions in an **initfuncs.c** file to perform board initialization prior to running a program.
- The **rcu** subdirectory contains an example showing how to load, patch and run a relocatable code unit (RCU) produced by the linker. The *ST20 Embedded Toolset Reference Manual* describes relocatable code.
- The **romdebug** subdirectory contains an example showing how to build and debug a ROM application; see the *ROM Systems* chapter in the *ST20 Embedded Toolset Reference Manual* for details.

Support material

The example directories **libs** and **mkf** contain support material relevant to more than one example. The **libs** directory contains libraries and the **mkf** directory contains makefiles.

The examples source utilize preprocessor defines for conditional compilation. For example the source file **examples/libs/led.c** uses the preprocessor symbol **ST_LED_FAMILY** to create a led flashing interface for a specific target board. If using a board configuration from the **board** directory then suitable values are defined for them using the **st20ccoptions** command.

7.4 The chip directory

The **include/chip** directory contains the header files that declare the register group base addresses for use by application code. These files, called address header files, declare address variables as **extern**, so the variables may be used in the same way as any other variable, and they are only instantiated if they are used. The address header files may be used by adding an appropriate **#include** directive to any C file, for example:

```
#include<chip/STi5500addr.h>
```

7.5 The source directory

The source code is held in subdirectories of the **src** directory.

Note: Subsequent toolset releases may invalidate changes that have been made to the source code in these subdirectories.

The **cstartup** directory contains the startup code for both linked unit and ROM loaded programs. These files are provided so that they can be modified to reduce overhead and provide additional initialization capability.

The **os20** directory contains the source code of the OS20 run time system. This is provided to enable customizing of the run time system for particular applications. Modifications to this software could invalidate the debug awareness capability of the toolset debugger.

The **traphandler** directory contains the source code for the traphandler, as used by the debugger. The traphandler includes code specific to particular chips. The source is provided to enable users to customize the traphandler for other chips if necessary.

7.6 The standard configuration file directory

The **stdcfg** directory contains some standard configuration files which define default definitions of targets and simulator resources and which are used by **st20cc** and **st20run**. These allow you to run the examples in the *ST20 Embedded Toolset User Manual* without first defining the characteristics of the target system.

It also includes the configuration files which describe each device specified by the **chip** command.

8 Release notes

8.1 ST Micro Connect 1

Depending when you received an ST Micro Connect 1, there may be newer usb and parallel port drivers.

To obtain the USB driver version currently used, make sure the ST Micro Connect is connected and from Windows select **System Properties**:

1. select **Hardware** tab,
2. select **Device Manager** then **ST Debug Interfaces**,
3. select **suitable HTI ?** entry,
4. select **Properties**,
5. select **Drivers** tab,
6. select **Driver details...**,
7. select **HTIUSB.SYS**,
8. **File version** identifies the version.

Update if **File version** is not 1.7 or later. Contact customer support or your FAE for the latest version.

The latest parallel port driver for Windows based systems **`par1284.sys`** is derived from the Warp Nine Engineering **`NTALL4.06`** release. This is the current shipping version from Warp Nine.

To check if this version is installed locate the **`system32\drivers`** directory.

For Windows XP this is **`C:\windows\system32\drivers`**.

For Windows 2000 it is **`C:\WINNT\system32\drivers`**.

Locate **`par1284.sys`** and right click. View **properties - version**. The version should be 4.06.0. If the version is lower, contact customer support or your FAE for the latest version.

Appendices

Referenced technology

A

The ST20 Toolset works in conjunction with other hardware and software technology, both ST and third-party. For convenience, details of all technology and versions mentioned in other parts of this manual are shown here.

The list of supported ST chips is shown in the ST20 Reference Manual, and, in the context of interrupt support libraries, in *Section A.2*.

A.1 Software and hardware platforms

Name	Version(s)[a]	Description
Microsoft Windows	2000, XP	Supported host operating system.
Red Hat Linux Enterprise	Workstation 3 or later	
Sun Solaris	SunOS 5.8 or later	

Table 4: ST20 Embedded Toolset referenced technology and versions

a. Other versions such as Windows ME may work, but have not been tested and are not supported.

A.2 Selecting interrupt libraries for ST chips

The *Interrupts* chapter of the *OS20 User Manual* describes how to link in different interrupt handling libraries for particular interrupt setups if not using the `chip` command.

B

Toolset revisions

This section describes the significant changes in the toolset functionality that have been made in each release since R1.6. Changes are categorized as follows:

- *General*
- *st20cc*,
- *C++*,
- *st20run*,
- *Command language*,
- *st20list*,
- *Debugger graphical interface*,
- *Simulator*,
- *EMI configurer stemi*,
- *Libraries*,
- *OS20*.

The file **`doc/buglist.htm`** is a list of known bugs that are yet to be resolved. It also lists the bugs resolved in this release since the previous release, (see *Section 7.2: The documents directory on page 24*).

B.1 General

B.1.1 Changes from R2.2 to R2.3

Supported Linux host o/s moved up from Red Hat Linux 7.2 to Red Hat Linux Enterprise Workstation Version 3.

B.1.2 Changes from R2.0 to R2.1

Removed **`stemi`**, an EMI configuration tool.

B.1.3 Changes from R1.9 to R2.0

New ST Visual Develop, **`st20dev`**, an integrated development system on Windows.

B.1.4 Changes from R1.8.1 to R1.9

The changes from R1.8.1 to R1.9 are:

- support for new silicon parts,
- support for Linux,
- new EMI configuration tool,
- POSIX I/O support,
- multi-threaded protection of the C runtime,
- time-logging support,
- new commands for ROM bootstrap delay and reading the toolset version,
- improved examples,
- bug fixes,
- major reformat of documentation suite.

B.2 st20cc

B.2.1 Changes from R2.2 to R2.3

New **st20cc** option **`-fas`** to align string literals to a word boundary for wide character support.

B.2.2 Changes from R2.1 to R2.2

- The linker command **`store -nocompact`** option is added to suppress zero data sequence compaction in a ROM segment.
- Linker runtime memory usage is improved in terms of the time it takes to link and to support linking larger applications before hitting *Windows Out of Memory* constraints on the PC.

B.2.3 Changes from R2.0 to R2.1

- C++ changes, see *Section B.3*,
- alterations to the TCOFF object and debug file formats to reduce file size,
  - ➢ faster compilation/linkage due to smaller files, particularly for C++,
- new feature to remove unused symbols,

- ➢ **st20cc** option **-remove-unused** instructs the linker to remove unused symbols so saving code and data space in the image,
- ➢ **st20cc** option **-resolve-locals** instructs the compiler to resolve local symbol references,
- ➢ linker command **keep** to identify symbols to preserve when **st20cc -remove-unused** option is used,

- compiler optimization to improve structure copy performance on ST20-C1.

B.2.4 Changes from R1.9 to R2.0

- C++ changes, see *Section B.3*,
- template placement is now possible using regular expressions.

B.2.5 Changes from R1.8.1 to R1.9

- linux hosts are supported,
- time-logging is supported, with the option **-debug-runtime**.

B.2.6 Changes from R1.8 to R1.8.1

- By default, **st20cc** now passes command line options to sub-tools (for example, the compiler and the linker) in the order that they appear on the **st20cc** command line. For backwards compatibility, the new **st20cc** option **-V18-cmdline-order** applies options in the order they would have been applied in earlier toolsets.
- Three other new options are provided as follows:
  - ➢ **-funroll-loops=*n*** to turn on loop unrolling,
  - ➢ **-place-exact** to force the linker to place sections in the order in which they appear in the configuration (**.cfg**) file,
  - ➢ **-warn-unused** to warn of unused variables and functions.
- **ST_device** may now apply to either a pointer or an object, which can be a **struct**.

B.2.7 Changes from R1.7 to R1.8

- New and modified command line options for C++ supersede those provided for earlier releases.
- Tentative variables have been implemented. These are essentially “common” variables shared between files. This does not create a back compatibility problem, but should be noted carefully as file scope variables not marked static will be potential candidates for “common”.
- **-wc** command line option suppresses warnings about nested comments.
- **st20cc** can now be configured by the end user to allow additional flags to be passed between the driver and the separate tools driven by **st20cc**. This is done using the **st20cc.cfg** configuration file. The new **st20cc** command line supports an option, as described in the *ST20 Embedded Toolset User Manual*, chapter *st20cc compile/link tool*.
- The file extensions associated with each tool may also be set in the **st20cc.cfg** configuration file. See the *ST20 Embedded Toolset User Manual*, chapter *st20cc compile/link tool*.

Other new command line options are listed in *Table 5* (see the *ST20 Embedded Toolset User Manual*, section 3.2.1).

The following **st20cc** command line options have been removed for the reasons given:

- **-t2**, **-t4**, **-t5**, **-w2**, **-w4**, **-w5**, **-E0**, **-E1 -GDUMP**, all **-PT***X* options (the C++ tools they apply to have been replaced),
- **-IFSD** (the **-H** option is preferred).

The **-EDU** option no longer applies to C++.

Option	Meaning
-H	Display file searching diagnostics for **#include** preprocessor directives.
-in-suffices filetype = [.ext1, .ext2 ..]	Allow one or more file extensions to be each associated with a file type, where the *filetype* may be one of those listed in *Table 6*. See also section 3.2.3 of the *ST20 Embedded Toolset User Manual*.
-quiet	Suppress close down messages from **st20cc**.
-CXX	Now deprecated, and tells user to use **in-suffices** instead.

Table 5: New st20cc command line options

File type	Meaning
cppfile	C++ source files
cfile	C source files
linkfile	Object files
libfile	Library files
asmfile	Assembler files

Table 6: in-suffices file type options

B.2.8 Changes from R1.6.2 to R1.7

Table 7 lists the new and changed command line options to **st20cc** from R1.6.2 to R1.7.

Option	Meaning
`-D`	Symbols with spaces are now supported.
`-depend file`	Generate makefile dependencies.
`-dl`	Create a RCU with a defined interface.
`-EDU`	Process `-D` option before `-U`.
`-IFSD`	Output diagnostics on compiler file inclusion.
`-lib`	Create a library.
`-make`	Do a `make` style date check to avoid recompilation.
`-makeinfo`	Display the reasoning behind the `make` process.
`-N`	Do not copy the debug information from the `tco` files into the `dbg` files.
`-nolibsearch`	Do not search the `ST20ROOT` area when linking.
`-nostdinc`	Do not compile with standard include files.
`-nostdlib`	Do not link against standard libraries.
`-PPE`	Preprocess source file with `#line` statements.
`-romimage`	Create a ROM image as output.
`-runtime`	Select a run-time system.
`-search_env {path}`	Override the `ST20ROOT` value.
`-suffix ext`	Define the extension used in object file names.

Table 7: New and changed st20cc options from R1.6.2 to R1.7

New **st20cc** options allow the target program to get segment and section information at run-time for use by `addressof`, `sizeof` and `sizeused`.

Table 8 lists the new and changed **st20cc** commands from R1.6.2 to R1.7.

Command	Meaning
`addressof` *`name`*	Return the address of a segment or section *name*.
`bootiptr` *`iptr`*	Define the initial IPTR value. This replaces `startstate`.
`chip` *`variant`*	Specify target ST20 variant.
`clerror` *`error`*	Report a command language error.
`clinfo` *`level`*	Define the level of progress information provided.
`clsymbol` *`symbol`*	Return the status of the symbol name.

Table 8: New st20cc commands from R1.6.2 to R1.7

Command	Meaning
define ***symbol_name value***	Define an integer constant with the given *value.*
endorder	Define the order in which shut down functions will be executed.
export ***symbol***	Support for exporting symbols.
import ***symbol***	Support for importing symbols.
memory ***seg addr size type***	New memory segment types.
mknum ***arg***	Convert a string to a number.
mkstr ***arg***	Convert a number to a string.
romimage	Specify the segments written into a ROM image file. This replaces **romfilename**, **romorigin**, and **romslice** commands.
sizeof ***name***	Return the size in bytes of a segment or section.
sizeused ***name***	Return the number of bytes used in a memory segment.
startorder	Define the order in which start up functions will be executed.

Table 8: New st20cc commands from R1.6.2 to R1.7

Other functional changes are as follows:

- When creating an RCU (relocatable code unit), a standard library is supplied in the linkage.
- When linking for C++, the standard C++ library is supplied in the linkage.
- Libraries created with the R1.6.x and OS-Link toolsets may be linked in with R1.7 code.
- A new **st20cc** directive **#ident** puts a string into the object file comment.
- Support within configuration files to allow start-up of OS20, rather than using calls in the user's program.
- Support for memory subsegments.
- This toolset can link in libraries and object files generated by the ST20-SWC toolset, as well as libraries and object files generated by the toolset, versions R1.6 and later.
- A new compiler built-in **__sizeofstructnopad** gives the size of a **struct** not including the trailing padding bytes.

B.2.9 Changes from R1.6.1 to R1.6.2

The driver and linker supported a new option **-N**. This directs the linker to not copy the debug information from the **.tco** files into the **.dbg** files. The debugger will attempt to find debug records in the **.tco** files if the **-N** option has been supplied to the linker. If the debugger cannot find a **.tco** file, then a warning will be displayed and the debugger will switch into assembly mode.

The linker is able to link libraries and object files produced using the OS-Link ST20 toolset. The debugger cannot access static variables that have been defined in a object file produced by the OS-Link toolset. Attempts to print such a variable will produce a variable optimized error.

B.2.10 Changes from R1.6 to R1.6.1

Support for packed structures was removed.

B.3 C++

B.3.1 Changes from R2.0 to R2.1

Faster C++ dependency file generation.

B.3.2 Changes from R1.9 to R2.0

The changes from, R1.9 to R2.0 are:

- C++ standard libraries (including STL),
- exception handling,
- templates,
- export templates,
- `#pragma` syntax,
- demangled C++ names may be used in all commands except `import` and `export`,
- better placement control of template code,
- thread-safe support of function-static class constructors and streams.

B.3.3 Changes from R1.8.1 to R1.9

Linux hosts are supported.

B.3.4 Changes from R1.8 to R1.8.1

- All C `#pragma` directives have been extended to C++.
- Separate C++ libraries are now provided which do or do not generate exceptions. The library version linked in is controlled by the `-exceptions` option to **st20cc**.

B.3.5 Changes from R1.7 to R1.8

- The C++ in R1.8 is a new implementation. The facilities provided and the **st20cc** options to build C++ code have changed substantially from earlier releases, and are described in the *ST20 Embedded Toolset User Manual*.
- C++ is provided in the R1.8 as a single tool **st20edg**. This tool uses the current standard method for template instantiation, as described in *ST20 Embedded Toolset Reference Manual, Part ANSI C and C++*. This replaces the tools **st20cfxx**, **st20gcpp**, **ptcomp**, **ptlink**, **tool1** and **tool2** for the preprocessing of C++ code in all earlier toolsets.
- The complex library is no longer included.

B.4 st20run

B.4.1 Changes from R2.2 to R2.3

Support for ST Micro Connect 2.

B.4.2 Changes from R2.1 to R2.2

- Fixed **`stderr`** as a separate output stream to **`stdout`**.
- Added the following sampled `IPTR` profiler enhancements:
  - ➢ output reports when the application contains too many functions,
  - ➢ correctly reports busy or idle transitions on the ST20-C1/DCU3 (ST20-C10x core).

B.4.3 Changes from R2.0 to R2.1

- Attempt a heuristic display of the callstack if the IPTR (program counter register) is corrupted.
- Extended the **`traphandler`** command with **`-static`** option to identify code linked into the program image to use for the debug support routines.This avoids **`st20run`** downloading this code at runtime into the **`TRAPHANDLER`** memory segment.
- Better enter include support when used with a GUI front-end.

B.4.4 Changes from R1.9 to R2.0

The changes from R1.9 to R2.0 are:

- C++ debug support,
- the UNIX **st20run** GUI is not available on Windows,
- trace facilities extended:
  - ➢ default trace configuration,
  - ➢ DCU3 fully supported,
  - ➢ postmortem trace extraction,
  - ➢ trace output control, for example: filters, views.
- breakpoint configuration extended:
  - ➢ non-trapping breakpoints,
  - ➢ extended sequencing options,
  - ➢ generate trace save event.

B.4.5 Changes from R1.8.1 to R1.9

- Support for the DCU3 on-chip diagnostic controller has been added, as described in the *ST20 Embedded Toolset User Manual*.
- Support has been withdrawn for JPI and B300 used together as an Ethernet host-target interface.

B.4.6 Changes from R1.8 to R1.8.1

- Support for ST Micro Connect host-target interface is added. See the *Interfacing to the target* chapter of the *ST20 Embedded Toolset User Manual*.

- Extension of the **chip** command to include new silicon. An improved and extended description of the **chip** command is included in the *Command Language* part of the *ST20 Embedded Toolset Reference Manual*.
- The debugger has improved awareness of programs when multiple programs exist on a ROM, for example when using OS20. The **program** command extensions can be used to aid the debugger, particularly for those programs which execute code from RAM. The debugger can handle C code built with toolsets R1.6.2 and later. See the *ROM Systems* chapter of the *ST20 Embedded Toolset Reference Manual*.
- **st20run** now automatically gets the debug information for the target I/O code. This means that there will be fewer occasions when the debugger enters assembly mode after an asynchronous **Stop**.
- The trigger support has been reinstated in the events handling (for example, sending a DCU trigger when a breakpoint is hit).

B.4.7 Changes from R1.7 to R1.8

- The debugger can single-step lines of source or single machine instructions (including changes of thread) using the **step** command, as described in the *ST20 Embedded Toolset Reference Manual*, or using the graphical interface, as described in the *ST20 Embedded Toolset User Manual*.
- The trigger support has been removed from the events handling (for example, sending a DCU trigger when a breakpoint is hit).
- Running the debugger on target boards with NOTRST disconnected is no longer supported. In versions of the toolset up to but not including R1.8, users were able to download and run programs onto target boards which did not have NOTRST connected. This was done by putting the target board into *boot from OS-Link* mode and resetting the target manually before connecting. **st20run** no longer supports this mode of operation. The NOTRST signal must be connected for **.lku** files to be downloaded and run.

B.4.8 Changes from R1.6.2 to R1.7

Table 9 shows the new and changed **st20run** command line options in this release.

Option	Meaning or change
-a ***arguments***	Pass ***arguments*** to the **main** function of the linked unit file.
filename**.dbg**	**.dbg** debug files can be supplied on the command line.

Table 9: New and changed st20run command line options in R1.7

Table 10 lists the new **st20run** commands.

Command	Meaning
bootiptr ***iptr***	Set the initial IPTR value, replacing the **startstate** command.
chip ***variant***	Specify target ST20 variant.
clerrror ***error***	Report a command language error.
clinfo ***level***	Define the level of progress information provided.

Table 10: New st20run commands in R1.7

Command	Meaning
`clsymbol` *`symbol`*	Return the status of the symbol name.
`go`	This command replaces the `run` and `continue` commands.
`informs -enable|-disable`	Turn debug input and output on or off.
`interrupts`	This command replaces the `action` command.
`timer`	Set up and manipulate debug timer events.
`traphandler`	Establish a trap handler.

Table 10: New st20run commands in R1.7

Table 11 lists the changed **st20run** commands.

Command	Change
`addressof` *`section`*	This command can now be applied to sections and segments.
`memory`	New memory segment types.
`register`	This command now supports the CPU group.
`session`	This command now saves the search path.
`sizeof`	This command can now be applied to sections and segments.

Table 11: Changed st20run commands in R1.7

Other functional changes are as follows:

- The way trap handlers are established on the target has been changed. Previous releases supported both the `DEBUG` segment and any linked-in trap handlers; this release only supports the `DEBUG` segment.
- Trap handlers for the STi5500 and ST20TP3 that keep the watchdog alive can be selected using the `chip` command.
- The debug trap handler is now loaded at run-time, rather than linked into the application program.
- `dbg` debug files are no longer required for IO or low-level debugging; they are needed only for symbolic support.
- `st20run` is not dependent on symbolic information for load, debug (when OS20 is not used) and I/O.
- The use of the debug library in RCUs is supported.
- Debugging of OS-Link linked-in libraries is supported, except for the printing of static variable values.
- Support for memory subsegments.
- Command language scripts are provided to list OS20 objects, in order to improve awareness of OS20 when debugging
- The Windows 95 and Windows NT EPP mode parallel port interface software has been improved.

B.4.9 Changes from R1.6.1 to R1.6.2

The debugger became able to handle multiple loads of a ROM images, as long as a **DEBUG** memory segment has been declared.

B.4.10 Changes from R1.6 to R1.6.1

- The support for debugging ROM loaded applications was improved.
- The breakpoint locating performance was improved.

B.5 Command language

B.5.1 Changes from R2.1 to R2.2

- Added chip support for the STB5188 and STB5107, updated STi5105.
- Removed **<ST20ROOT>/board** directory from the directory path added by the default **st20rc** file. This allows customers to use the same file names for their own board configurations as those in that directory.

B.5.2 Changes from R2.0 to R2.1

Added chip support for STi7710, STi5105 and STV0684.

B.5.3 Changes from R1.9 to R2.0

Demangled C++ names may be used in all commands except **import** and **export**.

B.5.4 Changes from R1.8.1 to R1.9

- The command **emi** has been added to support the new EMI configuration tool **stemi**, as described in the *Creating EMI configuration files* chapter of the *ST20 Embedded Toolset Reference Manual*.
- The file **chip.cfg**, supplied in the **stdcfg** directory, has been restructured.

  The **chip** command has support added for ST20DC2, STi5580, STi5519, STi5514, STi5516, STV0396 and STV3500.
- The **emidelay** command allows insertion of delays in a ROM bootstrap sequence.
- The **st20toolsetversion** command gets the version number of the current toolset.

B.5.5 Changes from R1.8 to R1.8.1

- A new command language procedure **cachecontrol** is introduced to control instruction and data caches. A description of the **cachecontrol** procedure is included in the *ST20 Embedded Toolset Reference Manual*, chapter *Alphabetical list of commands*.
- The commands **startorder** and **endorder** can now assign a priority to a function, which defines the order in which the functions are invoked on start-up and shut-down.
- **st20run** is now able to wait until a resource becomes available. The user can specify on the command line the number of retries and the wait between retries.
- The commands **write** and **mkstr** have new options to support a wider range of formatting.
- The new command **getenv** returns the value of an environment variable.

- The default value of the **tckdiv** parameter in the **target** command is changed from 1 to 4. See the *ST20 Embedded Toolset User Manual*, chapter *Interfacing to the target* chapter.

B.5.6 Changes from R1.7 to R1.8

- *Table 12* shows the new commands that have been added, as described in the *Command Language* part of the *ST20 Embedded Toolset Reference Manual*. Most of the commands were for handling files.

Command	Description
commandline	Tool configuration.
fclose	Closes open file.
feof	Tests for end-of-file marker.
fgets	Reads one line from a file.
fopen	Opens a file for reading, writing or appending.
fputs	Writes the specified string to a file.
rewind	Moves the file pointer to the start of the file.

Table 12: New file handling commands in R1.8

- The commands **fappend** and **fwrite** were deleted.
- The types of command language variables were extended by the addition of structures and arrays, as described in the *Command Language* part of the *ST20 Embedded Toolset Reference Manual*.
- The **tracebuffer** command now includes the option **-oneshot** to stop tracing when the buffer is full.
- A parameter **tckdiv** has been added to the target command. The clock speed of the JTAG interface is divided by the value of this parameter.

B.5.7 Changes from R1.6.2 to R1.7

The following changes in the built-in commands and language from R1.6.2 to R1.7 apply to all tools.

- The new built in command **clsymbol** tests whether a symbol is defined.
- The new built in commands **mknum** and **mkstr** convert a string to an integer and an integer to a string type respectively;
- New string operators, **+**, **-**, **==**, **!=**, **>=**, **<**, **<=**, **<<**, **>>**, **/**, **%**, **~**, **!**, *****, **&**, **|**, **+=**, **-=**, **>>=**, **<<=**, **/=**, **%=** and ***=**, have been added.
- The error behavior of the toolset command language has been enhanced, giving a back trace on error, and identifying the command in error.

B.5.8 Changes from R1.6.1 to R1.6.2

The **directory** command can be used to define alternative directories for the debugger to search.

B.6 st20list

B.6.1 Changes from R2.0 to R2.1

Decode ST20 RCU files (relocatable code unit).

B.6.2 Changes from R1.9 to R2.0

New options to list C++ symbols:

- `-NT` no truncate option,
- `-NDM` disable name demangling.

B.6.3 Changes from R1.7 to R1.8

In listings of TCOFF format binary files, names are now demangled.

B.7 ST Visual Develop, st20dev

B.7.1 Changes from R2.1 to R2.2 in st20dev

- Fixed the **Memory** display of negative address range.
- Added the **Clear** button to the **Trace** display to remove all trace records.
- Improved recursively adding source files to a project:
  - ➢ the build performance is improved when there are many hundreds of files,
  - ➢ there is more information in an error message to identify the source file names that clash.

B.7.2 Changes from R2.0 to R2.1

The changes from R2.0 to R2.1 are:

- Visual Studio.NET look and feel,
- new **Edit Event** dialog,
- new **Find in Files** dialog,
- trace navigation changes (scroll bar, paging),
- multiple Watch Tip formats,
- improved breakpoint resource management,
- **Go to line** dialog in editor,
- **Show/hide details** option on callstack window,
- ability to save and load trace data,
- ability to save and load core files,
- new **Registers** window,
- improved **Watch** window,
- editor file tabs,
- code breakpoints can be added and removed in edit mode,
- memory window contains **Find** box,
- memory window context menu has been reworked,

- syntax highlighting can be manually toggled on any file,
- **Rename** and **Delete** are available in Explorer window,
- **Hide full paths** option on disassembly window context menu.

B.7.3 Changes from R1.9 to R2.0

New in R2.0. Windows host only.

B.8 Debugger graphical interface

B.8.1 Changes from R1.9 to R2.0

- New tool - ST Visual Develop for Windows called **`st20dev`**. See *ST20 Embedded Toolset User Manual*. The UNIX GUI (**`st20run -g`**) is still supported with Solaris and Linux.
- The new trace features are only available through **st20dev**.
- Trace facilities have extended, in particular:
  - ➢ default trace configuration,
  - ➢ postmortem trace extraction,
  - ➢ trace output control: filters, views and other facilities,
  - ➢ DCU3 fully supported.
- Breakpoint configuration has been extended, in particular:
  - ➢ non-trapping breakpoints,
  - ➢ extended sequencing options,
  - ➢ generate trace save event.
- Change in the trace download behavior when using the UNIX GUI (**`st20run -g`**)
  - ➢ In pre-R2.0 toolsets, in **`Windows>Trace`**, once trace had been disabled then it would be extracted from the target and displayed. However, trace could only be sequenced on or off once using **`Windows>Trace Generation Commence`** and **`Halt`** selections.
  - ➢ In R2.0 the trace records are not extracted from the target until the **`Refresh`** button is selected. However, trace can be sequenced on and off multiple times using the commands **`break -traceon`** and **`break -traceoff`** in **`Window>Command Console`**.

B.8.2 Changes from R1.8.1 to R1.9

- Linux hosts are supported,
- RTOS data structures supported in the Variable window.

B.8.3 Changes from R1.8 to R1.8.1

TRIGGER-IN and TRIGGER-OUT have been reinstated.

B.8.4 Changes from R1.7 to R1.8

The **Step Line** operation has been added to the **Command** menu of the **Code** Window.

B.8.5 Changes from R1.6.2 to R1.7

- Buttons have been replaced by a toolbar.
- The values of registers can be modified and peripheral registers and groups can be defined.
- Memory values can now be modified.
- A window has been added to the PC version that displays the address map.

B.9 Simulator

B.9.1 Changes from R1.7 to R2.1

None.

B.9.2 Changes from R1.6.2 to R1.7

Table 13 shows the new and changed **st20sim** commands provided in the R1.7 release.

Command	Meaning
`addressof` *`section`*	This command can now be applied to sections and segments.
`bootiptr` *`iptr`*	Set the initial IPTR value.
`memory`	New memory segment types.
`sizeof`	This command can now be applied to sections and segments.

Table 13: New st20sim commands in R1.7

B.10 EMI configurer `stemi`

B.10.1 Changes from R2.0 to R2.1

Removed.

B.10.2 Changes from R1.9 to R2.0

None.

B.10.3 Changes in R1.9

This tool was added to R1.9, as described in the *ST20 Embedded Toolset Reference Manual*, chapter *Creating EMI configuration files*.

B.11 Libraries

B.11.1 Changes from R2.1 to R2.2

- A fix is added to ensure **errno** is thread-safe.
- The performance of library code is improved to handle double precision expressions on ST20-C1.

B.11.2 Changes from R2.0 to R2.1

- Performance improvements for ST20-C1 **memset**.
- Performance improvements for ST20-C1 in ROM bootstrap by using word copying.

B.11.3 Changes from R1.9 to R2.0

The C++ Standard Library is now provided.

B.11.4 Changes from R1.8.1 to R1.9

- The toolset now provides a POSIX-like I/O library for the target to read and write to arbitrary devices.
- Mutex management functions have been extended.

B.11.5 Changes from R1.8 to R1.8.1

- Header files are now provided defining the memory address variables used by the **chip** command. These files are in the **include/chip** directory of the release.
- The following library functions have been moved to different library files. This does not affect users who link using the standard supplied configuration files, but may require changes if the libraries are explicitly linked.

Functions	New library
setjmp, longjmp	**setjmp.lib**
The time manipulation functions **difftime**, **mktime**, **asctime**, **ctime**, **gmtime**, **localtime**	**ansigen.lib**
_ST_errno and errno	**stsyserr.lib**
All OS20 interrupt functions	**st20intc*.lib**

Table 14: New libraries

B.11.6 Changes from R1.7 to R1.8

The C++ complex library is no longer included.

B.11.7 Changes from R1.6.2 to R1.7

- Buffering has been introduced in the **stdio** C library implementation.
- Debug library routines can now return an error message that no debugger is present.
- A new **debugdisconnect** function terminates **st20run**, leaving the target running.
- A new support library for initializing RCUs.
- Library support for importing and exporting, to allow separately loaded programs to call each other.
- Support in the run-time library and start-up code for using subsets of the library.

B.12 OS20

B.12.1 Changes from R2.2 to R2.3

OS20 updated to version 2.11.06. Changes since 2.11.05:

- Time compare allows for equals as well as greater than a value (INSbl26941)
- Fix for failure to pre-empt low priority task when performing scheduling that interrupts **task_unlock** (INSbl27338)
- **timer_init_pwm** uses software to divide PWM4 ticks by 256 (INSbl26999)

B.12.2 Changes from R2.1 to R2.2

OS20 updated to version 2.11.05. Changes since 2.11.01:

- Object file **sti5100cache.tco** is available to work around problems with the data cache on some revisions of the STi5100. See *Appendix D.6: STi5100 cut3 and earlier on page 62*.
- Fixed a possible run queue corruption bug on the C1 (INSbl25657).
- Added some new chip names to **device_name**().
- Fixed **task_kill** status supplied to task exit handler.
- Fixed deadlock on ST20-C1 if semaphore is signalled during timeout.
- Fixed idle task incorrectly scheduled on ST20-C1 if scheduler locked.
- Fixed a thread-safety bug in **mutex_release_fif.o**
- Fixed a thread-safety bug in **kernel_reschedule** (both c1 and c2c4) that caused problems during the release of priority mutexes.
- Fixed mismanagement (incorrectly ignoring) the **start_next_task** bit in the c1 kernel.
- Fixed the c1 version of **task_context**() to correctly identify interrupt levels greater than seven.
- Removed two spurious asserts from the debug kernel.

B.12.3 Changes from R2.0 to R2.1

- Added support for 64 scheduling priorities on ST20-C1.
- Optional built-in ST20-C1 timer. Added:

  timer_init_pwm

  time_ticks_per_sec

  time_ticks_per_sec_set

This can be used instead of having your own ST20-C1 timer plugin.

B.12.4 Changes from R1.9 to R2.0

- C++ exception handling has been improved on ST20-C2 cores.
- Support for the mutex type has been added.

B.12.5 Changes from R1.8.1 to R1.9

- An interrupt level controller library **`ilc2b.lib`** has been added to support the STV0396.
- Time logging is supported in a new prebuilt OS20 debug library. Access to task and interrupt time logging information is provided by the existing **`interrupt_status`**, **`interrupt_status_number`** and **`task_status`** functions. System time logging information is provided by the new kernel functions **`kernel_idle`** and **`kernel_time`**. Time logging is described in the *Kernel* chapter of the *OS20 User Manual*.
- Default mutual exclusion is provided for the C libraries. This is installed by **`kernel_initialize()`**.

B.12.6 Changes from R1.8 to R1.8.1

- Support in OS20 for new on-chip interrupt controller and interrupt level controller hardware provided on new chip designs. Details of how to link in the correct interrupt library are given in section 18.2 of the *ST20 Embedded Toolset User Manual*.
- Several new OS20 functions are provided for cache control:

  `cache_config_data`
  `cache_config_instruction`
  `cache_disable_data`
  `cache_disable_instruction`
  `cache_enable_data`
  `cache_enable_instruction`
  `cache_flush_data`
  `cache_init_controller`
  `cache_invalidate_data`
  `cache_invalidate_instruction`
  `cache_lock`
  `cache_status`
- A new function **`partition_status`** returns the status of the heap, fixed and simple partitions. All partition functions return an error status if insufficient memory.
- New functions **`task_kill`**, **`task_mortal`** and **`task_immortal`** support the killing of a task.
- New functions are provided to support the managing of stack space, namely **`task_stack_fill`** and **`task_stack_fill_set`**. The new function **`task_status`** returns the status of a task, including stack details.
- The library filename **`os20errno.lib`** has been added to the configuration file **`os20libs.cfg`**.
- A new function **`task_resume_compatible`** has been added to provide behavior compatible with toolsets before release R1.7. See section *Section C.11: Task functions on page 57*.
- A documented function **`interrupt_status`** is provided in this release. This is not compatible with an undocumented version in toolsets earlier than R1.8.1, as an extra parameter has been added.

B.12.7 Changes from R1.7 to R1.8

None.

B.12.8 Changes from R1.6.2 to R1.7

- The following new functions have been added to the API:

  chan_create
  chan_create_address
  chan_delete
  chan_reset
  memory_allocate_clear
  partition_delete
  task_context

- For the API task and interrupt install functions, variations are now provided which take a function pointer as a parameter:

  interrupt_install_sl
  task_create_sl
  task_init_sl
  task_onexit_sl

  These allow the static link to be specified as well, which supports the use of RCUs with OS20.

- Callback is now supported by OS20. User-provided routines can be informed of scheduling operations, interrupt entry and exit, and task creation.

- Support on ST20-C2 cores for 2D block moves is provided by:

  move2d_all
  move2d_non_zero
  move2d_zero

- The amount of stack used by a task is increased by 2 words in the R1.7 release and later compared with R1.6.x. The algorithm for calculating stack usage given in the *ST20 Embedded Toolset User Manual* is unchanged, and is sufficiently generous to be enough for the new task usage.

  The amount of stack used by interrupts depends on the type of interrupt level controller the target device has. See the *Interrupts* chapter of the *OS20 User Manual* for more details of how to calculate stack size.

- If **task_reschedule** occurs within a task lock, it now does not cause the task lock to be lost. This is a change in the behavior of **task_reschedule** for release R1.7 and later, although the behavior in this circumstance was previously undocumented.

- From release R1.7, the number of nested task suspensions (that is, calls to **task_suspend**) is counted, so one **task_resume** is needed to undo each **task_suspend**.

- The behavior of OS20 functions **task_suspend()** and **task_resume()** has changed between ST20 Toolset R1.6.2 and R1.7, as described in section *Section C.11: Task functions on page 57*.

Porting code from earlier toolsets

C

The R1.7 version of the toolset introduced a number of new features, which require the user to make minor changes to their application or configuration files. This appendix describes these changes in a little more detail, to assist with the moving of applications developed with an earlier toolset to a later toolset than R1.7.

Appendix B: Toolset revisions gives a summary of all changes to the toolset; further details are available in the *ST20 Embedded Toolset User Manual*, the *OS20 User Manual* and the *ST20 Embedded Toolset Reference Manual*.

C.1 Object file compatibility

Object files (that is, `.tco` files and library files) have been generally backward compatible since version R1.5.1, and R1.5 if the object file does not explicitly place data in sections, with the exceptions described in the remainder of this chapter. Linked `.lku` files are not compatible, so any programs booted from the host using **st20run** will need to be linked again.

In this context, backward compatibility means that:

- `.tco` and library files can be safely linked without needing a new compilation,
- object files and libraries generated with old versions of the toolset can be linked by newer ones,
- object files and libraries generated by the newer toolset cannot be linked by older toolsets, except in the case of the R2.2 libraries and object files, which are compatible with R2.1.2-P2.

C.2 C++

See *Section 1.4: Comprehensive C++ support on page 3*, particularly the important notes.

C.3 Configuration file changes

Two changes to configuration files (used by **st20run)** are required when using the R1.7 and R1.8 releases of the toolset. (They do not apply to earlier versions).

- It is now mandatory to supply a **`TRAPHANDLER`** memory segment. This must be of type **`DEBUG`**, and be at least 1 Kbyte in size. Previously this was only required when debugging boot-from-ROM systems, but it is now always required.
- A **`chip`** command has been introduced, which can be used to describe the chip the program is running on. As well as defining the CPU type (ST20-C1 or ST20-C2) this also defines memory segments for internal memory and peripheral registers, and tells the debugger about many of the on-chip registers, so they can be displayed with the **`register`** command or in the register window.

For parts that have a DCU3, then the use of the **`chip`** command is mandatory. For instance, the STi5514, STV0396 and STV3500 have a DCU3 and so require the **`chip`** command.

For other chips, the use of the **`chip`** command is not mandatory (unless **`-runtime os20`** is used), but does simplify many configuration files.

When switching to using the **`chip`** command, it is usually necessary to remove existing definitions of internal memory and **`processor`** commands.

C.4 Incompatible changes

A few changes have been made to some library functions, which may break existing code, and require source code to be compiled again or modified:

- If your code explicitly uses the file descriptors **`stdout`** or **`stderr`**, then the code will have to be compiled again.
- The interface to the callback function used to protect **debug*()** functions from re-entrancy by multiple threads has changed. In previous versions the interface was:

  ```
  long int debugmutexfn(debugmutexop);
  ```

  However from release R1.7 this was changed to:

  ```
  long int debugmutexfn(debugmutexop, void*);
  #pragma ST_nolink(debugmutexfn)
  ```

The new second parameter to the callback function is set by the user using an extra parameter which has been added to **`debugsetmutexfn()`**.

This **`debugmutexfn`** has no access to global data, so if access to static data is required within **`debugmutexfn`**, then the static link (also known as GSB) must be supplied by the user data parameter. For an example of this see the OS20 source file **`$ST20ROOT/src/os20/clib/debprot.c`**.

C.5 Using -runtime os20

A new feature added in R1.7 was the ability to specify a run-time system when linking a program. The default run-time system is **c**, which provides a basic C run-time system. However OS20 can also be used as a run-time, in which case much of the initialization which a user had to perform in previous versions is performed automatically.

In general **-runtime os20** is most useful when developing new programs. However there are a few simple steps which can be used to convert existing code to use **-runtime os20**. Much of the initialization which is present in existing programs must be removed. In particular:

- calls to **interrupt_init_controller()**, **kernel_initialize()** and **kernel_start()** must be removed,
- any definitions and initialization of **internal_partition** and **system_partition** must be removed,
- any calls to **debugsetmutexfn()** to set up a debug mutex can be removed, as one will be installed automatically,
- any calls to **setstdiofiletablemutexfn()** and **setstdiomutexfn()** can be removed, as one will be installed automatically.

In addition the **chip** command must be used in the configuration file when linking the program.

C.6 RCU changes

The interface to relocatable code units (RCUs) has been changed. **st20cc** has the new commands **import** and **export** to allow separately loaded programs to call each other. **st20cc** has a new option **-dl** to create a RCU with a defined interface and **st20cc** supplies a standard RCU library in the linkage.

C.7 PrePoke and PostPoke callbacks in ROM bootstrap

The set of pokes encountered during linking is performed by the ROM bootstrap to support device initialization, such as initializing the EMI. To support device initialization that may require peeks as well as pokes, such as SDRAM, a user-supplied function may be called prior to these pokes, and a second user-supplied function may be called after these pokes. These functions are defined as follows and are linked in using an object file built from a user-supplied source file called **initfuncs.c** which includes the header file **initfuncs.h**:

```
void PrePokeLoopCallback(void);
void PostPokeLoopCallback(void);
```

C.8 Overlapping memory segments

The memory map of a target chip is provided by memory statements in a command file. This memory map is used by **st20cc** and **st20run**. To provide finer control of section placement, a memory segment can be declared which is within another memory segment. This subsegment must have a compatible memory type with the major segment it is within and it must not overlap the top or bottom of the major segment.

Table 15 is a summary of the memory types a segment can be split into. It is arranged with the major segment types in the columns and subsegment types in the rows. An x in a box indicates a compatible type, for example, a RAM segment can have a RAM, CODE or DATA subsegment.

Subsegment type	Segment type				
	RAM	ROM	CODE	DATA	CONST
RAM	x				
ROM		x			
CODE	x	x	x		
DATA	x			x	
CONST		x			x

Table 15: Memory segment split types

All the other memory types can be a subsegment of any other memory type, these are:

```
DEBUG
RESERVED
DEVICE
PERIPHERAL
```

A subsegment is treated by **st20cc** and **st20run** in the same way as any other memory segment so they can be used with the command language statements, for example, `place`, `addressof`, `sizeof` and `sizeused`.

Any command language references by **st20link** to a major segment take into account that it may have been split by subsegments. The address of the major segment is the first available part of the segment excluding any subsegments that are at the beginning of the segment. The major segment size is what remains after the subsegment sizes have been deducted. Placement into the major segment is located in the first part of what remains of that segment big enough for the placement.

C.9 Debug library status when st20run not attached

The debug functions that require **st20run** to service the request will return the status value `DEBUG_NOT_CONNECTED` if **st20run** is not connected. For example, `debugmessage` may return status value `DEBUG_NOT_CONNECTED`.

This is a change from R1.6.2 where the status value -1 is returned for this and also unspecified errors.

C.10 Standard error stream on PC Windows platforms

On PC Windows platforms, for toolset versions prior to R1.7, the output from the compiler, including warnings and errors, was sent to the stream called **stdout**. This stream could be redirected to a file or piped to the utility **more**.

Toolset version R1.7 sends warnings and errors to **stderr**. If there are many such messages, they can scroll off the top of the screen. On Win32, **stderr** can be redirected.

With version R1.8 on Win32, the default is to combine output directed at **stdout** and **stderr** into **stdout**, which is the same behavior as R1.6.

On Win32 it may be useful to send **stdout** and **stderr** to different files and this can be done using the option **-use-stderr**. This option is not available in UNIX. For example, to send all output to file **std.out**:

```
st20cc -p c2 demo.c > std.out
```

To separate **stdout** and **stderr** and write to two files **std.out** and **std.err**:

```
st20cc -p c2 -use-stderr demo.c > std.out 2> std.err
```

C.11 Task functions

If **task_reschedule** occurs within a task lock, it now does not cause the task lock to be lost. This is a change in the behavior of **task_reschedule** for release R1.7 and later, although the behavior in this circumstance was previously undocumented.

As from release R1.8.1, the behavior of toolsets before R1.7 can be reproduced by running the function **task_resume_compatible** with parameter **true**.

C.12 Multiple task suspension

The behavior of OS20 functions **task_suspend()** and **task_resume()** has changed. Suspensions of a single task may be nested so that the task may be suspended more than once. The same number of calls of **task_resume** is needed to resume the task.

The behavior prior to R1.7 can be invoked by the following call:

```
task_resume_compatible (1)
```

C.13 Parameter order

By default, from release R1.8 onwards, **st20cc** passes command line options to sub-tools (for example, the compiler and the linker) in the order that they appear on the **st20cc** command line. For backwards compatibility with toolsets before R1.8, the **st20cc** option **-V18-cmdline-order** applies options in the order they would have been applied in earlier toolsets.

Silicon bugs that affect the toolset

D

D.1 Introduction

This appendix lists the known silicon bugs that affect the working of the ST20 Embedded Toolset, and so are relevant to users of the toolset. Bugs that affect all hardware targets are listed in *Section D.2*. Bugs that affect some but not all target silicon are listed in the subsequent sections.

D.2 All chips with DCU2 diagnostic controller

D.2.1 Incorrect match on data breakpoints

DCU data valued breakpoints match on all address greater than or equal to the parameter value rather than only the value given.

Hence the command:

```
break var -h -d 24
```

will match on all writes to addresses that are greater or equal to **var** whose data value is 24.

D.2.2 Incoherent view of memory when both dcache and jumptrace enabled bug INSbl05750

When jump tracing is enabled all accesses to memory that are made using the DCU have the device bit set. When Dcache is enabled, the device bit means that CPU memory accesses are not cached. Consequently when both of these are enabled the DCU view of memory is not cache coherent. This has the effect that the jump trace may not contain a complete set of records.

Work-arounds

- Turn off the debug library using the **informs** command when jump tracing.
- Avoid mixing jump trace and viewing memory or variables.
- Do not use I/O (for example, **printf()**) when jump tracing.

D.2.3 DCU bug INSbl07059

There is a DCU bug whereby an access by the CPU to a DCU register may result in the wrong value being returned, depending on the memory accesses that may have preceded the DCU access. To work round this the following code should be executed prior to the DCU access:

```
ldlp 0; ldnl 0; adc 0; pop ;
```

For example, to read the DCU control register:

```
ldc 0x3004 ;
ldlp 0; ldnl 0; adc 0; pop ;
ldnl 0;
```

D.2.4 DCU bug INSbl20569

When trace is disabled with a '**break** ... **-traceoff -notrap**' command, jump trace cannot be re-enabled until the traphandler has been entered. This type of trace-off action may be required when the intention is to disable and re-enable trace so that only a chosen section of code is traced, or to miss out tracing of, for example, a **printf**. To work-around this problem, replace the **-notrap** option with **-continue**.

D.3 All HCMOS5 chips

D.3.1 Breakpoint traphandler re-entering

The breakpoint trap handler can re-enter if a DCU match condition (hardware code or data breakpoint) occurs while a software code breakpoint or a DCU data breakpoint is being handled.

Work-arounds

- Avoid setting breakpoints that cover the breakpoint trap handler workspace.
- Avoid writing breakpoint instructions into the breakpoint trap handler code.
- Avoid setting software breakpoints that occur immediately after a data breakpoint has occurred.

D.3.2 Incorrect jumptrace

If two jumps happen in quick succession, there is a possibility that the second jump is not recorded in the jumptrace buffer.

D.3.3 Ranged breakpoint problems

The comparison logic in break range is incorrect. If code is being executed in a break range, the current logic looks for a match between and including both the upper and lower address limits. This is incorrect, the logic should compare up to but not include the upper address limit.

On restart, from a breakpoint on the first location of a break range, the breakpoint trap handler will be re-entered.

If a ranged breakpoint is set on the instruction following a call to a subroutine, the breakpoint is sometimes missed on return from the call.

D.3.4 Breakpoint problem

Breakpoints do not work in certain circumstances related to a conditional jump instruction **cj**. The breakpoint will not work if there is a combination of:

- a breakpoint set on the target of a **cj**, and
- a breakpoint set on the instruction following the **cj**, and
- the jump is taken.

D.3.5 Traps and interrupts when processor is idle

On HCMOS5 versions of the ST20-C2 core, if the processor is idle and a trap or interrupt occurs and the processor returns from the trap or interrupt back to its idle state, then the DCU will become disabled.

The `stop` command cannot interrupt the target while the DCU is disabled, and therefore it has no effect in this case. The DCU is re-enabled when the processor becomes busy.

With toolsets R1.8.1 and later, there are software work-arounds in the breakpoint traphandler and the scheduler trap handler within OS20. Other traps and interrupts will still have the same effect.

D.4 HCMOS5 ST20-MC2

On the ST20-MC2 the breakpoint trap handler must be placed in internal memory.

D.5 ST20TP3 and ST20TP4

A problem is sometimes encountered with DMA engines being locked out. Sometimes a memory lock signal is left asserted after taking an interrupt, which can lock out DMA from peripherals. The lock can be freed by adding a device read to external memory at the start of all trap handlers.

Work-around

The OS20 interrupt handler code contains a work-around, which needs to read external memory.

OS20 users need to place a memory section named **`os20_lock_fix`** in external memory. Since the section is not written to, it may be placed in ROM.

The interrupt handler uses device instructions to read from this memory location, so it does not matter whether the location is cacheable or not.

D.6 STi5100 cut3 and earlier

The data cache must be flushed by software when it has been enabled. A read is required from each line to flush any dirty cache lines back to memory. Using the data cache controller register to flush does not work around the problem.

This release includes the object file **`sti5100cache.tco`** that contains replacements for the following OS20 functions needed for the STi5100:

- **`cache_enable_data`**
- **`cache_flush_data`**
- **`cache_invalidate_data`**

An application must be re-linked for the STi5100 with **`sti5100cache.tco`** as follows:

```
st20cc -p my_linkproc main.tco app.lib -runtime os20 sti5100cache.tco
```

D.7 ST20-C105 and ST20-C106 cores

Due to bug GNBvd42082, on ST20-C105 and ST20-C106 cores (with single cycle cache read) a CPU memory read of less than a word (for example, **lbinc**) may result in a word being read from memory. Normally this is alright as the CPU extracts the suitable byte from the word read. However, this will cause problems for memory systems that must only be byte accessed, such as that used by DVBCI.

The work around is to replace all **`char*`** or **short *** pointer de-references in the source with an assembly instruction sequence that precedes an **`lbinc`** or **`lsinc`** instruction with a **`ldnl`** instruction (to load the address of the access).

The following function has been validated to work around the problem:

```
#include "interrup.h"

unsigned char my_read8(unsigned char* src_p)
{
 unsigned char status;
 unsigned char** src_addr = &src_p;

  interrupt_lock(); /* ensure not descheduled between ldnl and lbinc */
  __optasm
  {
    ld src_addr;
    ldnl 0;
    lbinc;  /* ensure cpu issues lndl before lbinc */
    st status;
  }
  interrupt_unlock();
  return status;
}
```