vAccess™ User Manual

October 5, 2015

Table of Contents

Introduction to vAccess™ *3*

System Overview 4

vAccess Software Requirements 4

Getting Started *4*

Installing the Software 4

Using the Software 5

Installing a License 5

Using vAccess DLL *6*

ConnectVTOS 6

ExecuteCommand 7

DisconnectVTOS 9

Understanding vAccess.ini *10*

Valid INI File Settings 10

Example INI File 11

Troubleshooting *12*

Understanding the vAccess Program Files Directory *13*

vAccess Program Files 13

Understanding the vAccess Examples Directory *13*

vAccess Example Files 13

Running vAccessTest.exe *14*

Running vAccessTest.exe interactively 14

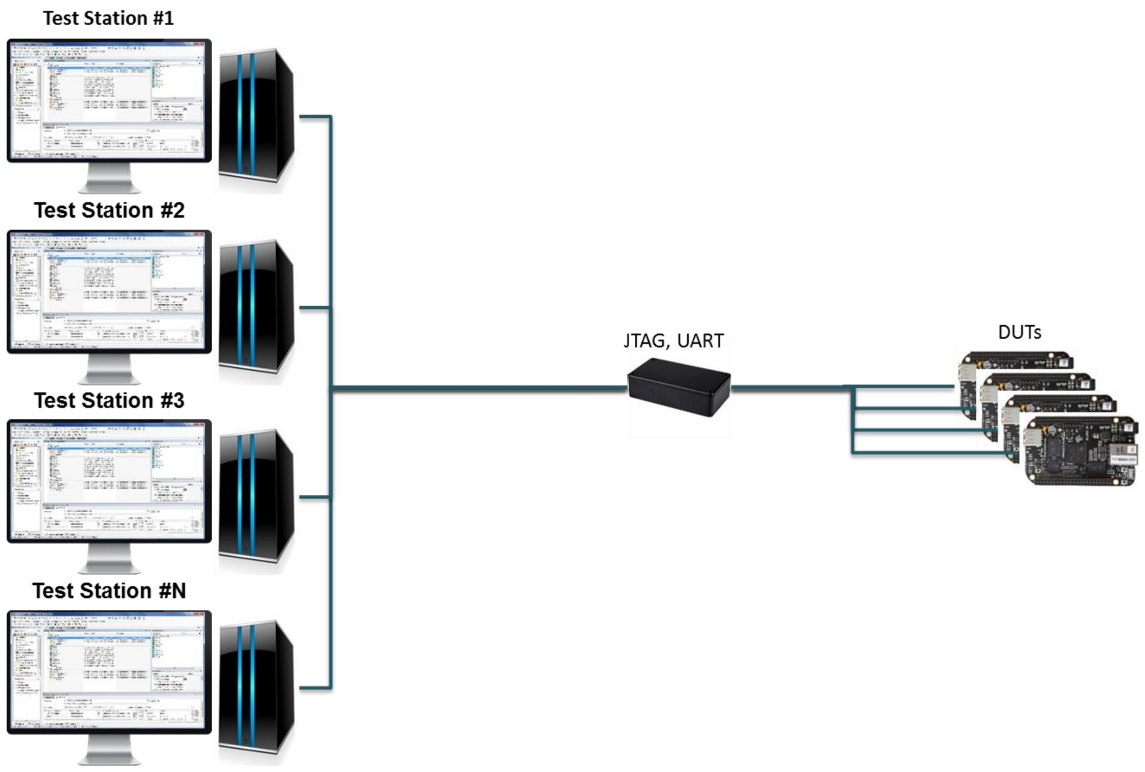
Running vAccessTest.exe programmatically 15

Contact Information *16*

About Kozio, Inc. *16*

Introduction to vAccess™

vAccess provides a DLL allowing third-party tools and products to communicate with VTOS DDR™, VTOS Program™, and VTOS Scan™ software running on an embedded device. vAccess provides a way to automate testing and embedded device programming.

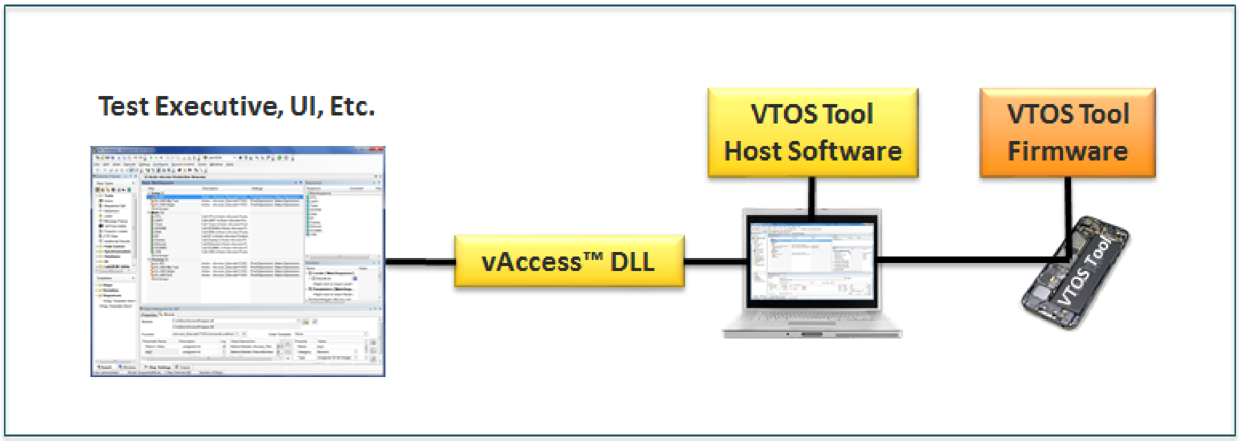


vAccess is a DLL that provides an API for integrating VTOS products with third-party tools, such as NI's TestStand, custom test executives, or third-party user interfaces.

All of the VTOS Tools (VTOS DDR, VTOS Program, VTOS Scan, etc.) are designed to work with vAccess. In fact, VTOS Tools can export project information that can be directly used by TestStand or custom test executives.

System Overview

The VTOS Tool system is comprised of two major components: VTOS Tool host software and VTOS Tool firmware. vAccess and VTOS Tool host software are provided as a single vAccess.dll. vAccess communicates with VTOS Tool firmware through a communication channel such as a serial port or JTAG hardware.



The VTOS Tool firmware runs entirely from the memory of your embedded system. The firmware provides a rich set of capabilities which are exposed through the vAccess API. The vAccess DLL runs on your PC. Each VTOS Tool user interface is described in its own user manual. This manual describes the vAccess DLL interface.

vAccess Software Requirements

Microsoft Windows XP or 7. Windows 8 has not been tested.
A serial port or other supported communication device is required for VTOS Tool use.

Getting Started

Installing the Software

vAccess is installed with an installer that includes all Kozio VTOS Tools. To install the vAccess software, run the latest VTOS Tools Installer. The installer is downloaded from www.kozio.com/support.

The installer creates a folder for the vAccess program files, an examples folder, and additional links under the Windows Start menu. The vAccess program files are stored in your Program Files directory and explained further in Understanding the vAccess Program Files Directory.

The examples folder is created in your User Profile directory. Provided are two examples: using vAccess with a NI TestStand Sequence and using vAccess with a custom C++ application. Each example demonstrates the main tasks required when using vAccess to execute VTOS Tool commands. See Understanding the vAccess Examples Files Directory for further details.

A single vAccess installation works with multiple VTOS Tools and all the embedded system architectures supported by those tools.

Using the Software

vAccess requires other software which provides the user interface. The vAccess DLL can be loaded by a custom test executive, commercial test executive such as TestStand, or by other third-party user interfaces. To use vAccess, you must integrate it with another user interface. This manual describes the vAccess API and how to properly use those API calls.

vAccess requires a license for full activation. Before you use the vAccess DLL for the first time, you must install a license file. A vAccess license file may be combined with other VTOS Tool licenses and all are contained in a single file with a ".lic" extension.

The vAccess software provides a programmatic interface to the VTOS Tool firmware running on your embedded device. This firmware must be loaded and running on your board using a JTAG programmer or an available boot method such as loading a boot image from an SD Card or USB interface. Instructions for installing and using the VTOS Tools are provided in their respective user manuals.

The vAccess interface also supports loading VTOS Tool firmware over a CodeWarrior TAP for a number of supported processor types. Please refer to the Kozio application note titled "Using VTOS Tools with CodeWarrior TAP". This application is available through the Kozio web site under application notes - http://www.kozio.com/support/application-notes.

Installing a License

vAccess requires a valid license for activation. A single license enables one instance of the software to be used on a specific host machine. Both node-locked and floating licenses are available. Contact sales@kozio.com to obtain a license for your use.

To install a node-locked license, copy the license file (*.lic) to your C:\ProgramData\Kozio directory. This folder is created after the installer runs and may be hidden, but can be accessed directly by typing "C:\ProgramData\Kozio" in the Windows file explorer.

To install a floating-license, copy the license file (Floating License.lic) to C:\ProgramData\Kozio. The floating license file can be created by anyone. It contains a single line "HOST 1.1.1.1" where 1.1.1.1 is replaced with the IP Address of the server running the Reprise floating license server. Download the floating license server and Instructions from www.kozio.com/support. Please see Application Note (AN134) Kozio License Instructions for further information.

Using vAccess DLL

The vAccess DLL software provides a programmatic interface to the VTOS Tool firmware running on your embedded device. Three main programming calls are required to use vAccess:

1. Connect to a VTOS Tool
2. Execute a Command
3. Disconnect from the VTOS Tool

These function prototypes are provided in vAccess.h. This file can be found in the %KOZIO_VTOS_TOOLS% folder. This program header file also contains values for return codes and other functions useful for automating VTOS Tool usage.

ConnectVTOS

This function call is used to establish an inter-process communications channel to a particular VTOS Tool. This call must be made successfully before any VTOS Tool commands can be executed.

This function requires three input arguments:

- ***INIFile*** is a character string with the full path to a vAccess.ini file. See Understanding vAccess.ini for further details.
- ***Fixture*** is an integer representing the communication channel to be used for communicating with the VTOS Tool firmware running on the embedded device. Valid Fixture numbers are defined in the INI file.

The ConnectVTOS call returns a status value which is one of the vAccess return codes defined in vAccess.h. Refer to vAccess.h for the full list of potential error codes.

- When successful, VACCESS_SUCCESS is returned.
- VACCESS_CONNECTION_IN_USE is returned when this connection is already in use.
- VACCESS_STATUS_NO_LICENSE is returned when a license file could not be found in the User profile directory.
- VACCESS_CONNECTION_FAILURE is returned when vAccess failed to establish an inter-process communication channel with the VTOS Tool host software.

ExecuteCommand

This function call is used to execute a VTOS Tool command. Possible VTOS Tool commands are defined in separate API documents.

This function requires five input arguments:

- ***Command*** is a character string containing the command to be executed. A simple example is the command string "version".
- ***RetStr*** is a pointer to memory allocated for text data returned while executing the command. This text buffer may be filled with output from the command executed. See Understanding Console Data for further details.
- **MaxStrSize** is the size in bytes of the text data buffer (RetStr).
- ***RetStrSize*** is a pointer to an integer holding the size in bytes of the text data written to RetStr. This value will indicate how many bytes were written to RetStr.
- ***TestStatus*** is a pointer to an integer holding the status of a test command when executed. This value will hold one of the vAccess test return codes values.

A command is determined to be successful when it is received and processed by the VTOS Tool firmware. When you execute a command for testing, scanning, or programming, that command processing may be successful, but the TestStatus variable indicates whether the test passed or failed.

The ExecuteCommand call returns a status value which is one of the vAccess return codes defined in vAccess.h. Refer to vAccess.h for the full list of potential error codes.

- When successful, VACCESS_SUCCESS is returned.
- VACCESS_NOT_CONNECTED is returned when ConnectVTOS has not been successfully executed.
- VACCESS_EXECUTE_COMMAND_FAILURE is returned when vAccess was unable to send the command to the VTOS Tool firmware.
- VACCESS_EXECUTE_COMMAND_ABORT is returned when the command was accepted by the VTOS Tool firmware, but took too long to complete.
- VACCESS_EXECUTE_COMMAND_TIMEOUT is returned when the command was accepted by the VTOS Tool firmware, but we did not receive a command completion status before vAccess timed out.

Understanding Console Data

In most cases, executing a VTOS Tool command will result in extra information being displayed during the execution of that command. When using a VTOS Tool in an interactive mode, that extra information is displayed to a console window. When using vAccess, that information is captured in a return buffer.

RetStr is the text buffer containing any extra information that is captured during the execution of a VTOS Tool command. The parameter MaxStrSize tells vAccess the maximum number of characters to store, and RetStrSize is returned with the actual number of characters stored.

When you execute a command and it is successful, the extra console data is not required, but may prove useful for further details. When a command fails, the extra console data is valuable for understanding the details of the failure.

The vAccess API provides two additional calls for working with console data.

- ***int ClearConsoleBuffer()*** will clear the circular buffer storing all console data. The buffer is automatically cleared after and buffer read operation.

- ***int ReadConsoleBuffer(char *RetStr, unsigned int MaxStrSize, unsigned int *RetStrSize)*** - will retrieve all captured console data and store it in RetStr. MaxStrSize indicates the maximum size of the return buffer and RetStrSize indicates how many bytes were returned. This function returns immediately and uses RetStrSize to indicate if any data was retrieved.

Understanding TestStatus

The TestStatus parameter is used to return pass, fail, or abort status from a test command.

A test command is a special class of command that returns additional information. The "version" command is not a test command, but the "test.sdram.walk0" is a test command. When you execute a command, and it is not a test command, TestStatus returns with a value of VACCESS_TEST_PASSED. A test command will return a vAccess test return code.

- When a test passes, TestStatus is set to VACCESS_TEST_PASSED
- When a test fails, TestStatus is set to VACCESS_TEST_FAILED. In this case, you can make additional calls to return all of the error codes that were recorded. See Retrieving Error Codes for further details. Also, the RetStr text buffer may provide additional useful information.
- When a test is aborted, TestStatus is set to VACCESS_TEST_ABORTED. This will happen when vAccess executes a test and it takes longer than expected to complete. Also, the RetStr text buffer may provide additional useful information.

A test suite is a special command that contains many individual test commands. An example is "test.ddr.structural". The return code in this case is a bit ambiguous, you will know if all tests passed, or one or more test commands failed or aborted.

- When a test suite passes, TestStatus is set to VACCESS_TEST_PASSED

- When a test suite has one or more test commands that abort or fail, TestStatus is set to the relevant error code. In this case, you can make additional calls to return all of the error codes that were recorded. See Retrieving Error Codes for further details. Also, the RetStr text buffer may provide additional useful information.

Retrieving Error Codes

When a test command executes and fails, it records an error code. The error code provides additional information useful for debugging the error or determining root cause. A single test can produce many error codes.

The vAccess API provide two additional calls for clearing and returning error codes.

- ***int ClearErrorCodes()*** will clear the circular buffer storing all error codes.

- ***int ReadErrorCode(unsigned int *ErrorCode)*** - will look up and store an error code in ErrorCode and return VACCESS_SUCCESS. If no error codes are stored, VACCESS_NO_ERROR_CODE is returned and ErrorCode is zero. If a test fails, call this function multiple times until VACCESS_NO_ERROR_CODE is returned.

SetCommandTimeout

This function call is used to adjust the amount of time to wait before a command is aborted.

This function requires one argument:

- Milliseconds - The amount of time to wait before timing out an executing command.

The SetCommandTimeout call returns a status value which is one of the vAccess return codes defined in vAccess.h. Refer to vAccess.h for the full list of potential error codes.

- When successful, VACCESS_SUCCESS is returned.

DisconnectVTOS

This function call is used to disconnect from the VTOS Tool and clean up all resources used. This call must be made successfully before trying another ConnectVTOS command.

This function requires no arguments.

The DisconnectVTOS call returns a status value which is one of the vAccess return codes defined in vAccess.h. Refer to vAccess.h for the full list of potential error codes.

- When successful, VACCESS_SUCCESS is returned.

Understanding vAccess.ini

vAccess relies on an initialization file to define one or more fixtures. A fixture is a shortcut notation for defining a communications channel between the VTOS Tool host software and VTOS Tool firmware.

Note: The vAccess.ini file is automatically updated by the VTOS Program and VTOS Scan user interfaces. It is best to use those GUIs for modifying this file. To update vAccess.ini, simply launch the GUI and select File -> Connections or the Connections icon in the tool bar. This file is shared by all VTOS Tools.

When you establish a new connection between vAccess and a VTOS Tool, you specify the INI file and fixture number. vAccess uses the information from the initialization file to properly establish the connection between the host and embedded device, using the specified fixture number. You must configure at least one Fixture in the vAccess.ini file.

The initialization file adheres to Microsoft Windows INI file standards. Sections are indicated with '[ ]', and contain keys of the form key[=value]. Each section configures the parameters for a particular fixture. The valid keys are given below.

Valid INI File Settings

[General]	This is the first section of the INI file. It contains values common to all fixtures.
Fixtures=Integer	This key defines the number of fixtures and is required and must match the actual number of [Fixture#] defined.
Debug=Integer	This is an optional key. It defines the verbosity of debug output. 1=Always (default) 2=Low 3=Medium 4=High
Timestamp=Integer	This is an optional key. It defines whether to display (1) timestamps in the log files or not (0).
[Fixture#]	This label defines a new section for a new Fixture. The actual label is Fixture1, Fixture2, etc. Define at least one Fixture.
VTOSFirmware=String	This is the full path to the VTOS Tool firmware object file. Environment variables may be used for the path name.
VTOSHomePath=String	This is the full path to the VTOS Tool home directory. Environment variables may be used for the path name.

FixturePort=String	This key defines the communication channel used by this fixture along with the details needed for that channel. Several communications are planned. Here are the currently supported communications channels: ● comm-serial.dll, COM#, baud_rate, data_bits, parity bit, stop bit

Example INI File

```
[General]
Fixtures=3

[Fixture1]
VTOSFirmware=%KOZIO_USER_HOME%\VTOS_DDR\vtos.sitara_am335_ddr.elf
VTOSHomePath=%KOZIO_USER_HOME%\VTOS_DDR
FixturePort=comm-serial.dll,COM72,115200,8,none,1

[Fixture2]
VTOSFirmware=%KOZIO_USER_HOME%\VTOS_Scan\vtos.sitara_am335_scan.elf
VTOSHomePath=%KOZIO_USER_HOME%\VTOS_Scan
FixturePort=comm-serial.dll,COM6,115200,8,none,1

[Fixture3]
VTOSFirmware=%KOZIO_USER_HOME%\VTOS_DDR\vtos.sitara_am335_ddr.elf
VTOSHomePath=%KOZIO_USER_HOME%\VTOS_DDR
FixturePort=comm-serial.dll,COM92,115200,8,none,1
```

Troubleshooting

If you are having trouble establishing a connection or executing commands, the first place to look is in the vAccess log file. The log file is created and updated each time vAccess runs.

The vAccess log file is located in the User Profile logs folder. This folder is located here: C:\Users\USERNAME\Kozio\logs\, where USERNAME and drive letter vary for each user. The installer created environment variable %KOZIO_USER_HOME% also points to this folder.

Tip: Modify the vAccess.ini file and use the Debug=Integer key to increase the information reported by vAccess. See Understanding vAccess.ini. Start with Debug=2 to see additional information, then use Debug=3, and only use Debug=4 if the other levels did not help.

Tip: Look in the other log files to determine if any errors occurred.

Here are some common problems to look into.

VACCESS_STATUS_NO_LICENSE	• This error can occur if vAccess is unable to locate the *.lic file, or can't talk to the license server. The debug log will contain INI parsing information and other information which will help determine root cause. • This error can occur if you have multiple DLL's running and you only purchased one license. Use the Windows Task Manager to make sure you don't have extra or stranded Kozio processes running. Kozio's processes have "vtos" or "vaccess" in their process name.
VACCESS_CONNECTION_FAILURE	• This error can occur when you have incorrect information or paths specified in the vAccess.ini file. Double check the INI file and also look at the vAccess log information. Use Debug=2 to see INI information.
VACCESS_EXECUTE_COMMAND_ABORT	• This error will occur when the DUT is offline or not communicating over the Fixture communications channel.

Understanding the vAccess Program Files Directory

The installer creates a vAccess program files directory in your Program Files directory. The directory contains the vAccess software files. The location of this directory is stored in the %KOZIO_VTOS_TOOLS% environment variable.

vAccess Program Files

vAccess.dll	vAccess DLL object file.
vAccess.h	vAccess C program header file.

Understanding the vAccess Examples Directory

The installer creates a vAccess folder in your User Profile directory. The folder contains an NI TestStand sequence file and two C++ test applications. The location of this directory is stored in the %KOZIO_USER_HOME% environment variable.

vAccess Example Files

Kozio-vAccess-DDR.seq	This NI TestStand Sequence file provides an example of using the vAccess DLL to execute VTOS DDR™ commands. Please refer to the ReadMe.txt for additional information. **Note:** The call to *Connect VTOS DDR* and *Load Board Configuration* must be changed since it specifies a path to the INI file and this varies with each user. **Note**: This example calls more DDR test commands than would be necessary for a true production test run.
Kozio-vAccess-DDR_Report[12 22 53 PM][5 26 2015].xml	This is an example NI TestStand report file that was created using the above sequence file. Open this file in Internet Explorer to view what your report should look like when running the example *.seq file.
mingw-build.bat	This batch file builds the example C++ application using MinGW. MinGW has to be installed in order for this to work.

ReadMe.txt	This text file provides additional information about the examples providing in this folder.
vAccess.ini	This is an example vAccess initialization file using by the sample C++ test application and the sample TestStand sequence file.
vAccessTest.cpp	This is an example C++ application using vAccess to execute any commands.
vAccessTestDDR.vcxproj	This is a Visual Studio 2013 project file for building the C++ test application. Please refer to the ReadMe.txt for additional information.
vAccessTest.vcxproj	This is a Visual Studio 2013 project file for building the C++ test application. Please refer to the ReadMe.txt for additional information.

Running vAccessTest.exe

vAccessTest.exe is a pre-built executable that allows you to make vAccess DLL interactively or programmatically calls.

Running vAccessTest.exe interactively

To run vAccessTest.exe interactively, just launch a cmd.exe window and traverse to the %KOZIO_VTOS_TOOLS% directory. The command “cd %KOZIO_VTOS_TOOLS” will get you to the right location. Run vAccessTest.exe and you should see a prompt. At the prompt, you can type 'help' for usage.

The typical sequence needed to run test commands to connect, run commands, and then disconnect. Once connected, you can run as many commands as desired. Here is an example of running a VTOS DDR test sequence:

```
vAccessTest> `connect C:\Users\JoeS\Kozio\vAccess\examples\vAccess.ini 1
vAccessTest> version
vAccessTest> dut.version
vAccessTest> include
C:\\Users\\USERNAME\\Kozio\\VTOS_DDR\\boards\\sitara_am335\\beaglebone
_timing_based.ksc
vAccessTest> ddr.configure
vAccessTest> test.ddr.structural
```

vAccess test uses the character (`) as an escape sequence to tell vAccessTest to do special setup. All commands without this character are sent to VTOS Tool firmware for processing.

Running vAccessTest.exe programmatically

To run vAccessTest.exe programmatically, you pass it three arguments to specify the INI file, fixture number, and Kozio script file.

```
vAccessTest Command Line Usage:

vAccessTest [-i IniFile -f Fixture# -r ScriptFile -v -t -d milliseconds]
Use -i to specify the full path to the vAccess.ini file (required)
Use -f to specify the fixture number (required)
Use -r to specify the full path to the Kozio script file (*.ksc) (required)
Use -v to specify verbose output (optional)
Use -t to print command timing (optional)
Use -d to delay N milliseconds after connecting to the target (optional)
```

To repeat the last example, using a script file, we create a text file with the following lines in it. For this example, the file is labeled “ddr-test.ksc”.

```
version
dut.version
include C:\\Users\\USERNAME\\Kozio\\VTOS_DDR\\boards\\sitara_am335\\beaglebone
_timing_based.ksc
ddr.configure
test.ddr.structural
```

To run the last example, we execute from a command window:

```
vAccessTest –i C:\vAccessTest\vAccess.ini –f 3 –r ddr-test.ksc
```

The source code for this example is provided and available for your own internal use. Using the command line option to run script files allows you to quickly create a primitive test executive for running regression tests or temperature chamber testing.

Contact Information

For technical support questions, please email support@kozio.com.
For sales or other information, Kozio, Inc., +1 (303) 776-1356 x1, sales@kozio.com, www.kozio.com

About Kozio, Inc.

Kozio, Inc. is a software technology company focused on providing superior embedded tools solving a variety of challenges during the design, production, and support of embedded devices. With its wide range of embedded tools, Kozio offers solutions to aerospace, automotive, consumer, industrial, military, medical, networking, and wireless markets. Kozio has been crafting embedded software since 2003 and has served the needs of thousands of engineers working for hundreds of companies, from the smallest to the largest.
Kozio's line of embedded tools include everything you need to configure, test, and tune DDR memory; identify interconnect faults and component placement problems on your printed circuit board; and program on-board devices such as NAND Flash, NOR Flash, eMMC, FPGAs, and other programmable devices. Kozio's tools are reusable across a family of SoC-based designs and reusable across teams