Test automation with MDK and ULINK*plus*

AN 307, January 2018, V 1.0

arm KEIL

feedback@keil.com

Abstract

Modern application development is not a single-user task anymore. Large teams – often covering various time zones – require that the application software must be tested fully automated regularly to let the engineering management check the progress daily. But not only large teams have adapted a workflow that uses test automation and continuous integration, also smaller teams are using build servers to run nightly build tests of their embedded project.

This application note shows how to use debug scripts to:

- communicate with the target under test using the I/Os on the ULINK*plus*
- emulate user input
- change the program flow for test purposes
- read and write memory as well as core registers
- check the states of microcontroller I/Os.

This document is the first in a series of technical papers that will help you to understand the needs of modern application development. The series explains how these needs can be fulfilled with MDK and the new debug adapter, ULINK*plus*.

Contents

Abstract 1

Introduction 1

Test example 2

  Step 1: Connect the hardware 2

  Step 2: Download the project and build the code 3

  Step 3: Run the test in batch mode 3

Test details 3

  Debug procedure based on Test.ini 3

  Troubleshooting 4

Appendix 5

Introduction

This application note explains how to run a project fully automated using the µVision command line and debug script engine, and how to use ULINK*plus* for simulated user interaction.

Prerequisites

To run through the material, the following software and hardware is required:

- A Keil MCBSTM32F400 development board (but any development board will do if you have access to some of the I/Os and an on-board 10-pin Arm Cortex debug connector)
- A ULINK*plus* debug adapter
- MDK v5.25 pre-release (www.keil.com/mdk5/525) or newer

Copyright © 2017 Arm Ltd. All rights reserved

Test example

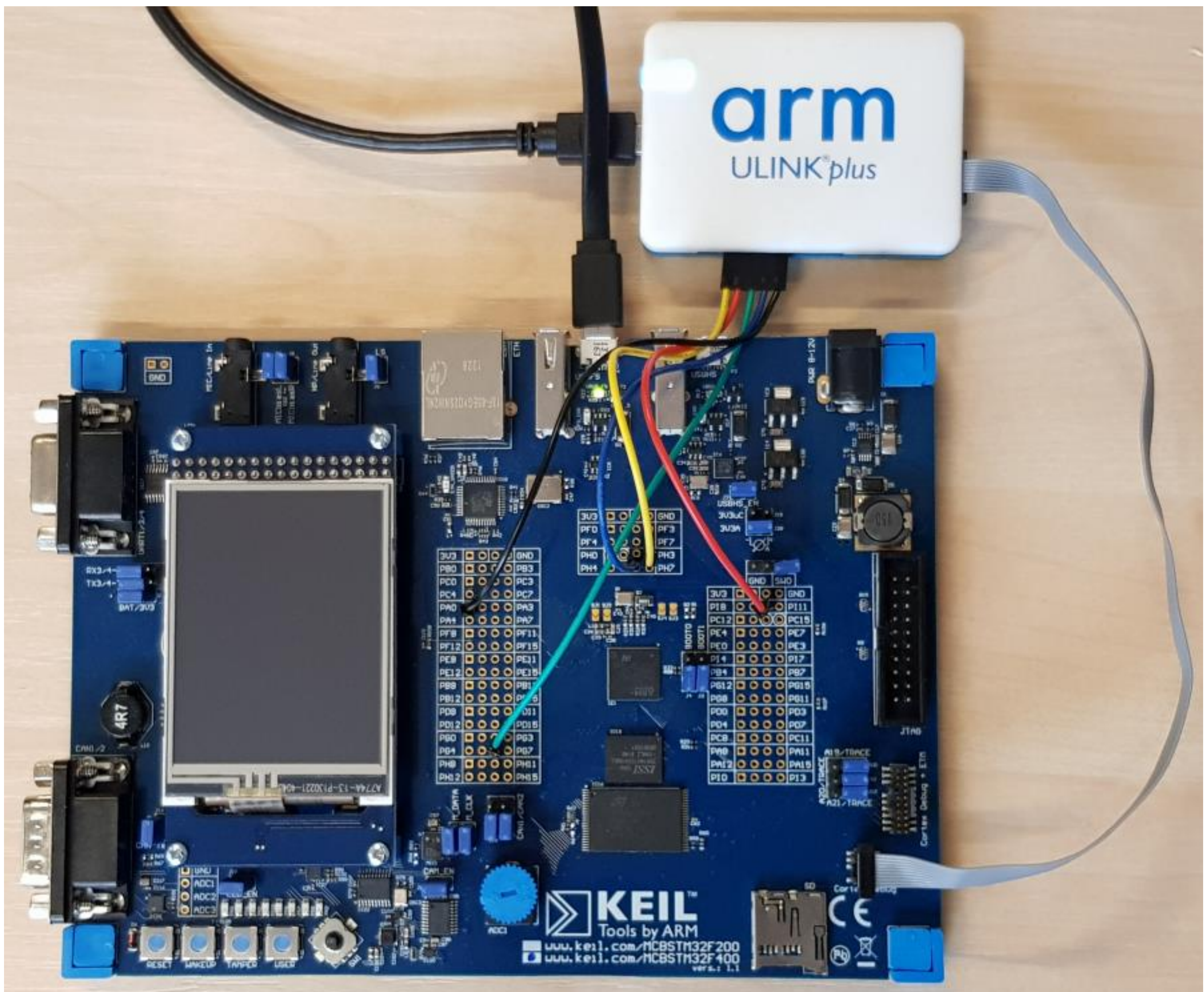
The example project runs several tests to verify the correct functionality of the processor, hardware, and software. For further information on the tests that are executed, refer to the “Test details” chapter. To setup and run the test, follow these three steps:

Step 1: Connect the hardware

Attach the ULINK*plus* to the 10-pin **Cortex Debug** connector on the target board. Use jumper wires to connect the ULINK*plus* I/Os to the development board I/Os:

ULINK*plus*	MCBSTM32F400	Board function
IO0	PH3	LED0
IO1	PI10	LED3
IO2	PG6	LED4
IO3	PH2	LED7
IO4	PA0	WAKEUP button

Connect the ULINK*plus* and the MCBSTM32F400 development board to your computer using Micro-USB cables.



Copyright © 2018 Arm Ltd. All rights reserved

Step 2: Download the project and build the code

Download the project from www.keil.com/appnotes/docs/apnt_307.asp and open it. It is a simple project using one thread to light up the eight LEDs on the MCBSTM32F400 development board. It is using Keil RTX5 as the underlying real-time operating system.

Build and flash the project to verify the correct behavior. You should see the eight LEDs below the LCD screen blinking.

Step 3: Run the test in batch mode

The project contains a batch file, called `Test.bat` that runs the automated test and checks the results. To run the test from the Windows command line, enter the following in a **Command Prompt** window:

```
C:\> Test.bat C:\Keil_v5\UV\UV4.exe
```

Adjust the path to your MDK installation if necessary. If you are using this standard installation path, you can omit it and just call `Test.bat`.

The batch file first builds the application, then flashes it onto the target hardware, and finally runs the debug session. To verify the proper operation, it checks for the availability of the test results and prints them onto the command line (see Figure 1).

Note: all calls of µVision are using the `-j0` option that suppresses the GUI. This is called headless mode which is especially important for automated building and testing as usually no one is checking the graphical output.

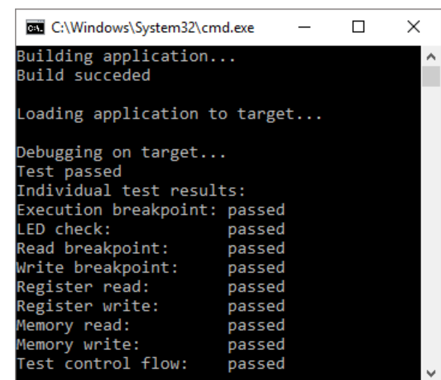


Figure 1 Command window output

Test details

In the project, two debug scripts are used to run the automated test:

1. `Debug_UlinkPlus.ini`: this file is available as a template from your MDK installation (refer to the Appendix). Using this file, you can configure the I/O pins and enable power measurement. Here, it sets the I/Os and includes the `Test.ini` debug script. When the test has finished, it exits the debug session.
2. `Test.ini`: This file is used to run the application specific tests.

Debug procedure based on Test.ini

In the `Test.ini` file, the actual debug procedure starts at line 487 by removing all previously set breakpoints, so that the following test can run undisturbed. Then, it calls the function `OpenLog` to start logging the output of the script.

Execution breakpoint

At line 493, an execution breakpoint is set on the line with the `LED_On` statement in `main.c`. Execution breakpoints halt the program execution or execute a command when the specified code address is reached. For every `BreakSet` command, you can specify the number of counts that determines the number of times a breakpoint condition is met before the target program halts or the specified command is executed.

To be able to check whether the number of times is correct, the variable *bpExecCounter* is added to the C code. The code execution continues until it hits the breakpoint. The function `CheckBpExec` checks whether *bpTestCounter* has the right value.

Copyright © 2018 Arm Ltd. All rights reserved

Using ULINKplus to enter a special test mode

To change the execution flow, a signal is sent to the application by setting I/O0 of the ULINK*plus* to high. The C code sets the LEDs 0, 3, 4, and 7 to signal the test mode. The `CheckLEDs` function compares the ULINK*plus* inputs against the expected values.

Read access breakpoint

A breakpoint is set to the variable *test_success* to monitor read accesses to this symbol. The code execution is continued. The function `CheckBpRead` checks if the variable is read at the right program counter using the helper variable *bpReadCounter*.

Memory write test

The function `MemWrite` fills the array *test_array1* with test data (256 values; incremented from 0x1000). In the application, this data is copied to *test_array2* and later verified for correctness.

Register read and write test

The function `RegReadWrite` reads and writes the internal registers of the Cortex-M4. First, the internal registers are read and it is checked that they are read correctly. In a second step, they are written and the content is verified.

Write access breakpoint

A write access breakpoint is set to the variable *test_success* to monitor accesses to this symbol. Eight single steps are executed to step over the first occurrence of a write access to *test_success*. Register R6 is modified to set the number of elements that are copied in the program from one array to the other to the right value.

The code execution is continued. The function `CheckBpWrite` checks if the variable is written at the right program counter using the helper variable *bpWriteCounter*.

Memory write test

The function `MemRead` validates the data in *test_array2*. If all previous operations have been successful, the data should be the same as the one that has been written to *test_array1* previously.

Test evaluation

Finally, the success of all tests is evaluated and printed out on the command line and into a separate `Test_results.txt` file. In a continuous integration environment, you can use this file to measure the success of your build test. Use the `EvalSuccess` function to adapt the text file's output to your needs.

Troubleshooting

If your test run does not pass, you can start a debug session without the script by commenting out these lines in the UlinkPlus_Debug.ini script:

```
// INCLUDE .\Test.ini
…
// EXIT
```

When entering debug, you then stop at main.

Copyright © 2018 Arm Ltd. All rights reserved

Appendix

ULINK*plus* introduction videos

A set of videos showing how to use ULINK*plus* is available here: www.keil.com/ulink/plus

ULINK*plus* user's guide

For more information, visit www.keil.com/support/man/docs/ulinkplus

Debug_UlinkPlus.ini template debug script

This template debug script is shipped with µVision. It contains methods to set up the ULINK*plus* I/Os and to enable the power measurement, as well as functions that you can use to generate analog or digital signal patterns on the ULINK*plus* outputs. This script is available from the [Installation_directory]\ARM\ULINK\Templates.

µVision Command Line

Invoke µVision from a command line to build a project, start the debugger, or download a program to Flash. The command applies to project and multiple-project files.

For more information, visit www.keil.com/support/man/docs/uv4/uv4_commandline.htm

Debug Commands

Debug Commands can be used in the µVision **Command** window and in debug functions. Debug commands can be built using expressions that include numbers, debug objects, operands, program variables (symbols), fully qualified symbols, system variables, or virtual registers (VTREGs).

For more information, visit: www.keil.com/support/man/docs/uv4/uv4_debug_commands.htm

Expressions

Many debug commands accept numeric expressions as parameters. A numeric expression is a number or a complex expression that contains numbers, debug objects, or operands.

For more information, visit www.keil.com/support/man/docs/uv4/uv4_db_expressions.htm

ULINKplus Virtual Registers (VTREGs)

The ULINK*plus* I/O pins can be accessed using Virtual Simulation Registers, or VTREGs. This makes them usable for test automation by scripting logic digital levels or applying/measuring analog signals.

For more information, visit www.keil.com/support/man/docs/ulinkplus/ulinkplus_using_ios.htm

Copyright © 2018 Arm Ltd. All rights reserved