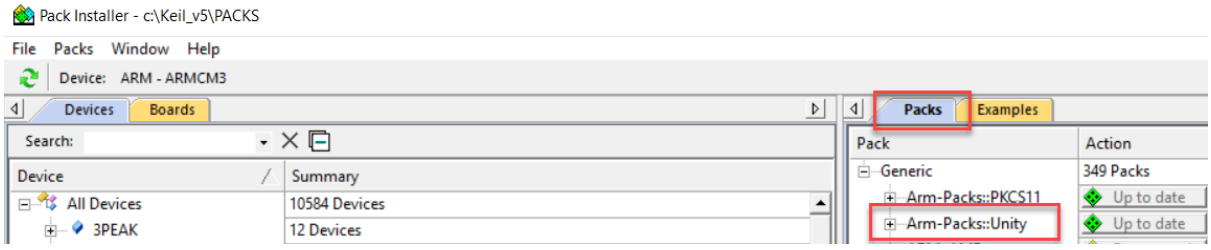
Unity Software Testing Framework

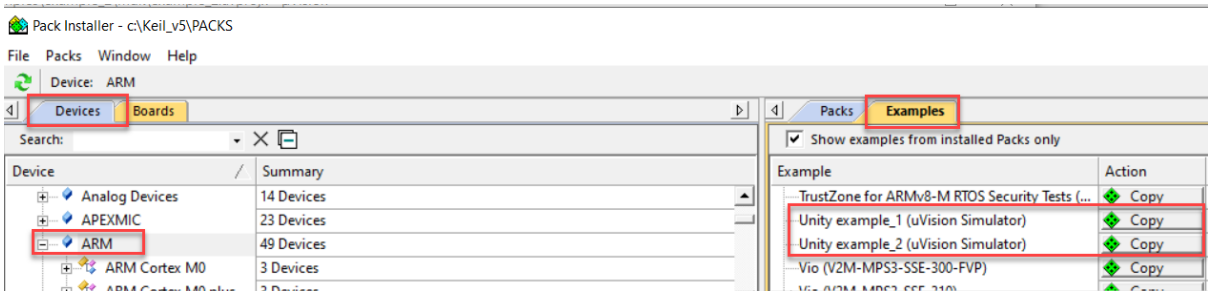
In this tech tip we will look at how to set up a project to use Unity, a popular testing framework for small embedded systems. Unity is written for use with the C language and can be easily added to most projects. It is designed to have a small footprint so can be used with most microcontroller projects.

Install Unity

Unity is part of the CMSIS pack system and can be added to the RTE through the pack installer.



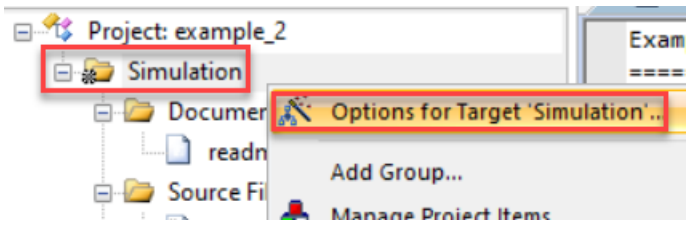
Once installed there are two example projects that demonstrate how to use unity within a microvision project. These can be found by selecting ‘Arm’ as the device vendor and then selecting the examples tab. Once located press the copy button to open the example.



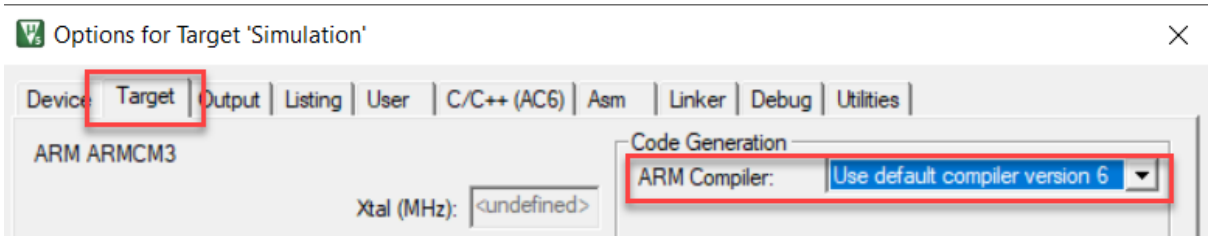
These examples are a good starting point as they demonstrate both a minimal framework and a more advanced test fixture. However, these examples use the version 5 compiler which has been removed from the current version of MDK. So we first need to update them to use the Version 6 compiler.

Converting to Arm compiler Version 6

Right-click on the project root and select options for the target.



In the target menu select “Use default compiler 6”



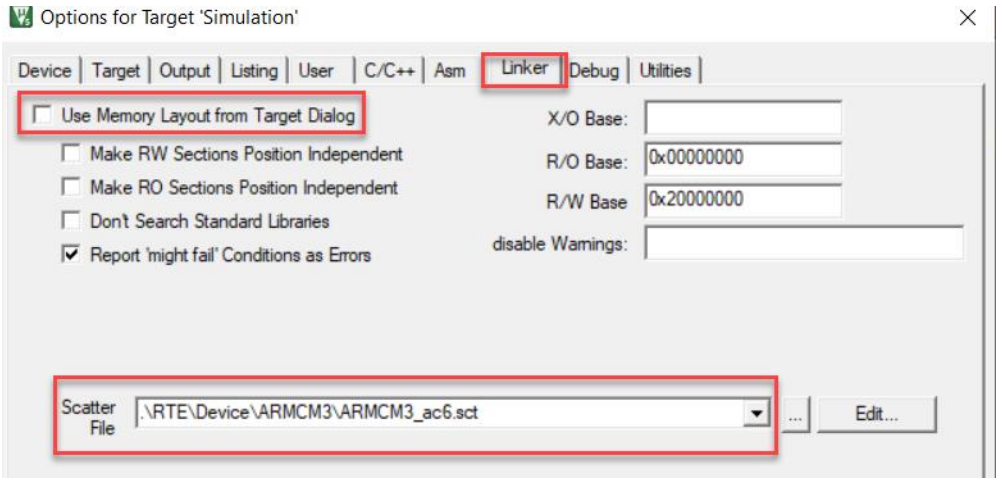
Close the options for target and then reopen it again

This will update the project files to the version 6 compiler

Select the C/C++ tab and remove the options in the misc controls box

These will cause errors when used with Version 6

Now select the linker tab



Uncheck the Use memory layout fromTarget dialog

Select the ARMCM3_ac6.sct scatter file in the <project>\mdk\RTE\Device\ARMCM3

Click ok

Now the project should build as normal

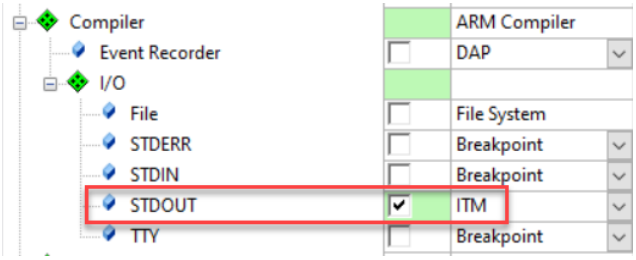
Adding Unity to your Project

In both projects the unity framework is added in the test leaf within the RTE Manager

This adds the full test framework to your project.

Redirect STDIO to the Instrumentation Trace ITM

The results of each test case are printed in a report format (Text or HTML) to the standard IO channel. The instrumentation trace is an ideal channel to use so the test results will be displayed in the console window of the debugger. However, the smaller Cortex-M processors (M0 and M0+) have a minimal debug architecture which does not include an ITM. In this case it is possible to use the event recorder in place of the ITM. This will require more on-chip RAM, if this is a problem you would need to use a physical channel such as a UART.



Example1 demonstrates a minimal framework that runs a collection of test cases.

Example2 adds a more sophisticated test fixture that can be used to manage a large number of test cases in the form of test groups.

Each test is created using a set of Asserts, a typical test is shown below.

```
TEST_ASSERT_EQUAL(0, FindFunction_WhichIsBroken(78));
```

There is a large range of test asserts and documentation can be found by clicking the blue link in the description column within the RTE.

In both examples, we are testing individual modules of code and the test framework provides its own main() function.

Both examples make use of the legacy simulator in Microvision so it is possible to build and run the projects in the debugger.

A good development approach is to create a 'sub-project' for each module or associated collection of modules (aka software component). You can extensively test the code before integrating it with the main project.

For more information visit our website: www.hitex.co.uk or get in touch: info@hitex.co.uk. You can also connect with us: @hitexuk (Facebook & X)