Starting a new project in Keil µVision 5

By Colin Funnell

Introduction

After a number of years using Keil µVision for a range of Arm Cortex-based designs, I've boiled-down various notes into some useful tips. As obvious as some of them sound, project pressures sometimes try to force a jump ahead and miss some of the more useful things. Getting a platform ready from the start instead of trying to enable features later will save a few headaches.

Some tips crop-up from experience with a number of projects. Others come from leveraging the power of the µVision debugger for trickier bugs. Getting it all set-up at the start will make your design all the easier to work with. Let's kick-off with…

Use a development kit close to your expected platform

Software can get a jump-start before available hardware by using the same expected micro-controller. A dev kit with similar I/O is better again. Quite a few dev kits are specifically supported and are a handy comparison if your product hardware has a mysterious bug. Dev kits with embedded debugger circuitry may not be as full featured as a dedicated debugger with full connectivity.

Choose an MCU with good CMSIS driver support

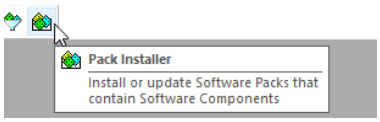
Common Microcontroller Software Interface Standard (CMSIS) is an excellent way to access common MCU features and middleware such as UARTs, SPI, I^2C up to WiFi. Common drivers allow you to re-target your design if needs be. Not all devices have good CMSIS support from their vendors. You can see devices catered for and their level of CMSIS support at www.keil.com/dd2.

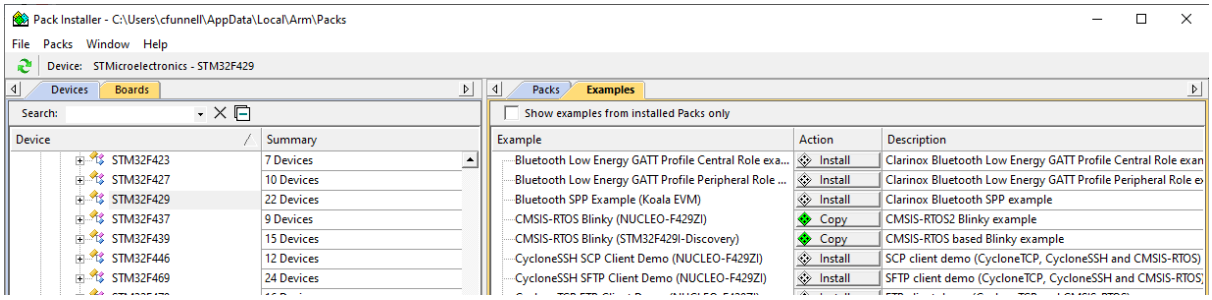
Understand the target device

The Arm Cortex-M family certainly helps set the general expectations of the device, but there is still a range to consider. The Cortex-M specifications allow a number of features to be optional. Devices are available which go from the smallest and lightweight to the large, heavy-weight beasts of the MCU world. Silicon vendors also include their own special features which help set them apart from the competition. Never assume different devices of the same M-class are the same. Some vendors can even supply assisting tools and code packages which others do not.

Start with an example project

These can be found in the Pack Installer. After selecting a device, the support packs are listed under the "Packs" tab and suitable examples under the "Examples" tab. The example project should be enough to at least generate and debug code. A pain-free way to start your project work without worrying about subtle configurations to stop you before you even get going. Some examples for certain devices are of their time and might not show the best starting point. Modern silicon and their dev kits are better than an old favourite for this. Don't forget to rename and regroup your design from the example files. You don't want yet another project still called "Blinky"! (It does give the game away somewhat.)

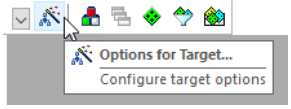


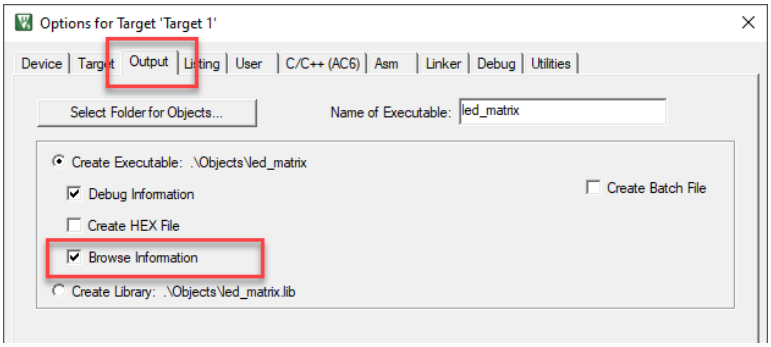


Enable Browse Information

For the editor to help support the developer, additional information must be created along with the build in general. I wouldn't live without it, but it isn't the most obvious option to find.

Under Target Options, Output tab, tick the Browse Information box. Debug Information should already be ticked.





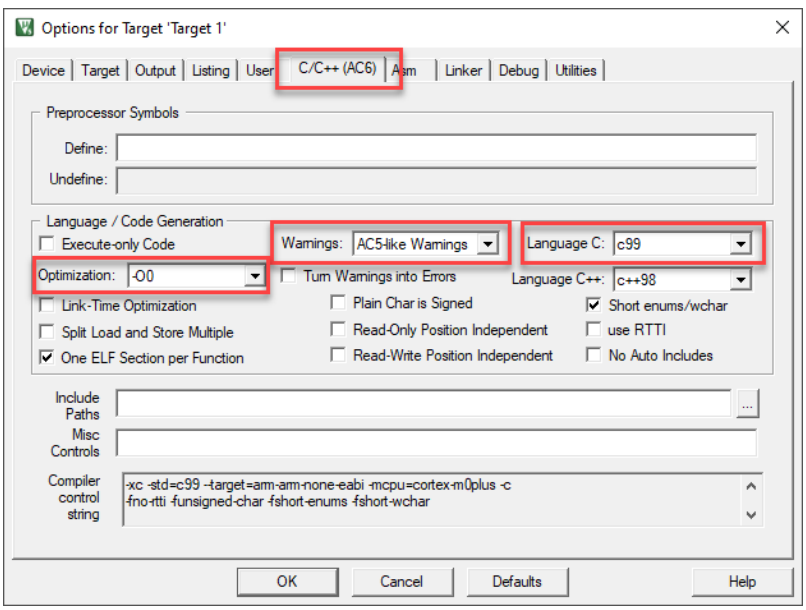
Choose compiler options suited to the code packs

Especially useful if starting from an empty project configuration, default code blocks expect to be compiled by a particular compiler. With the many options available it isn't unheard of to have a wave of warnings crop up. As a starting point we want to see zero errors and warnings so we can spot our own problems. Currently, most code compiles with the C99 language variant with AC5-like warnings enabled. For new projects make sure compiler version 6 is used.

Change code optimisation to none (level 0)

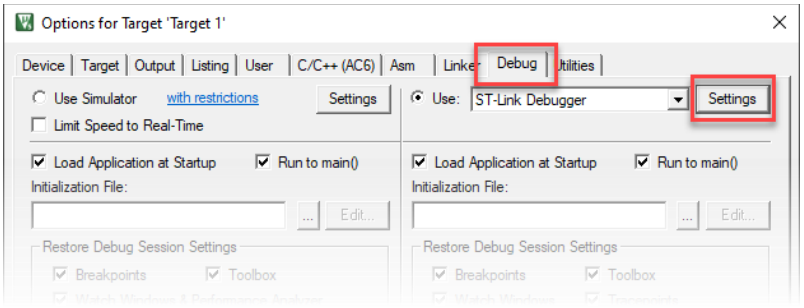
Some starting projects have code optimisation enabled to some degree. With optimisation enabled, single-stepping through code may appear to jump around – making it harder to follow an elusive bug and certain breakpoints disallowed. With no optimisation (-O0) there is no arguing what you're seeing and its ordering.

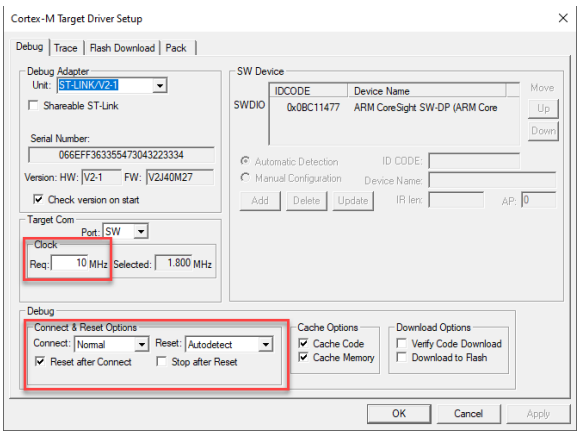
There is the added bonus that if a project starts to run out of processing power or memory space as it progresses, you now have an option up your sleeve to quickly optimise your way out of it for the moment.



If initial debug connections go awry

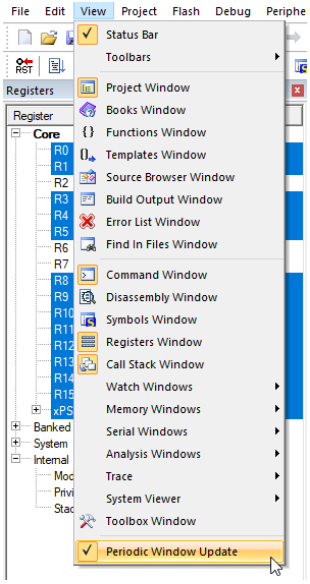
Adjust some of the defaults of the debug connection. Slowing the debug port clock down and adjusting the reset conditions during connection may uncover unusual startup problems. Disabling debugger code and memory caches will impact performance but will also reduce the doubt of any mis-matches between tool and target.





Make sure debugger Periodic Window Update is enabled

There have been a number of times I have had running code but no obvious update of internal variables. Save yourself some head scratching by checking the obvious - that variables in the debugger will be shown with updated data! This option is selectable from within the debugger window once a debug session has started.

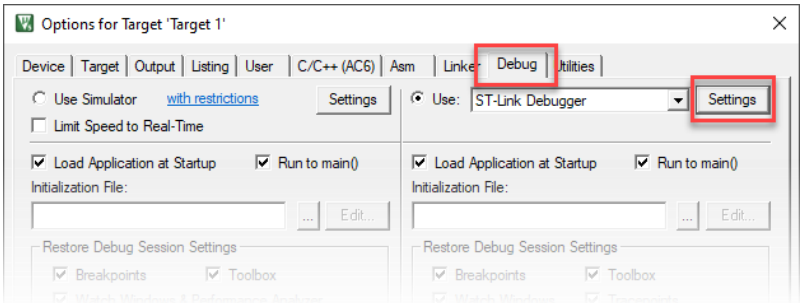


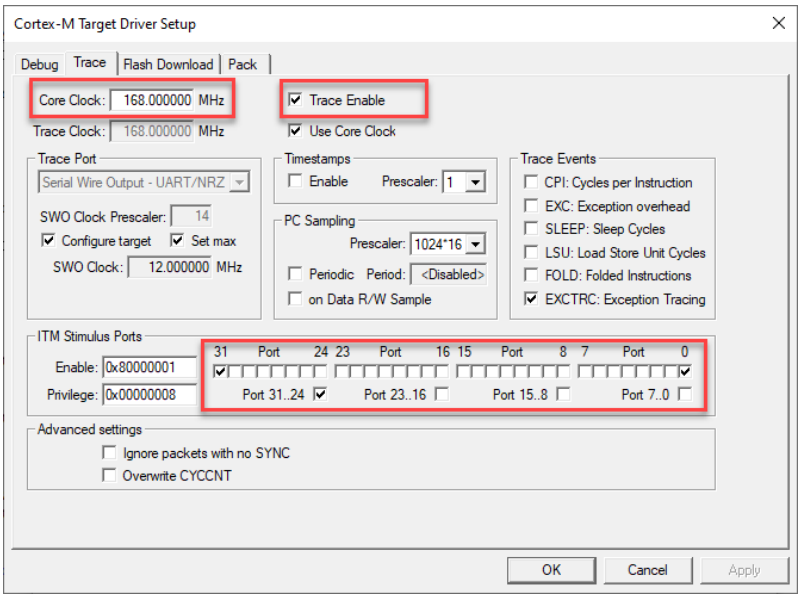
Ensure the debugger has the correct core clock frequency

This won't stop basic debugging but can throw measurements and synchronisation out on the more advanced features. It is best to get this set early in the project to avoid annoying snags just when you need it. This should be shown in the startup code or configuration tool for your MCU.

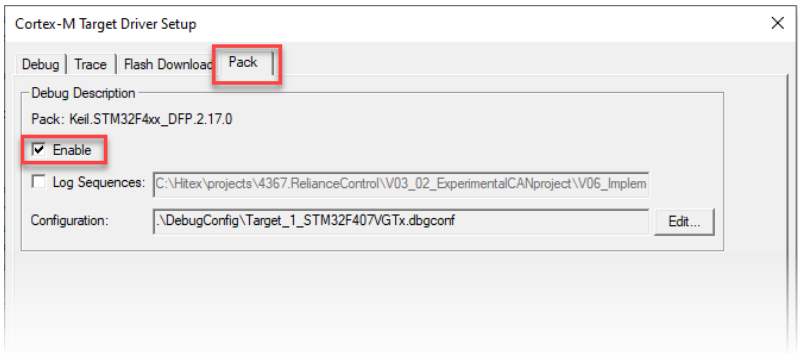
Enable debugger use of ITM (where available)

The optional Instrumentation Trace Macrocell inside an MCU allows a handy mechanism to allow some basic I/O through the µVision debugger serial window. It provides a quick and easy way to interact with the target using printf(), scanf() and fgetc(). It won't work without the correct debugger clock settings and isn't available on Cortex-M0(+).



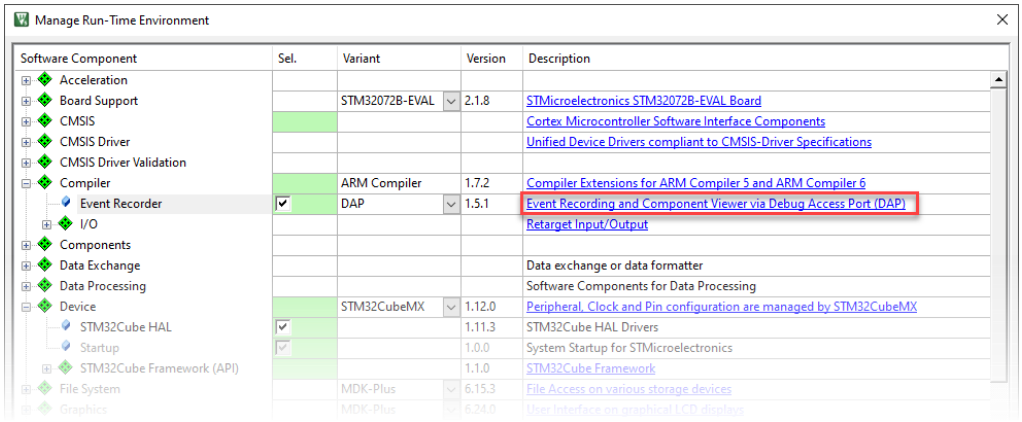


The pack tab should also have a custom debug configuration enabled for the microcontroller. This is used to enable MCU specific features such as changing a particular pin from GPIO to SWO. None of the debugger trace features will work if this isn't done.



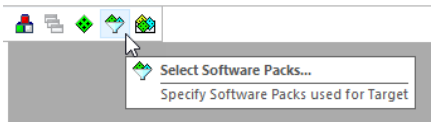
Enable the use of Event Recorder

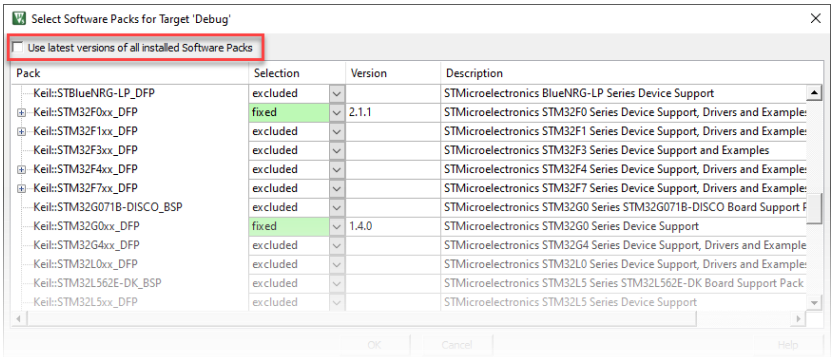
This becomes more involved from the developer's point of view but provides a rich source of traceable information. There is too much to quickly show here, but it offers an insight into a time-wise list of events in the system by sacrificing a small amount of memory to log it in. The Keil RTX5 RTOS outputs to it but it is open to pretty much any code adding information into it to help show and time events. It is quite an invaluable tool for a real-time system and has been a big debug enhancement on M0 devices. The µVision System Analyzer window is basically a graphical representation of the Event Recorder data. See the Run Time Environment / Compiler / Event Recorder help for details on how to configure and use this.



Disable the use of latest software packs

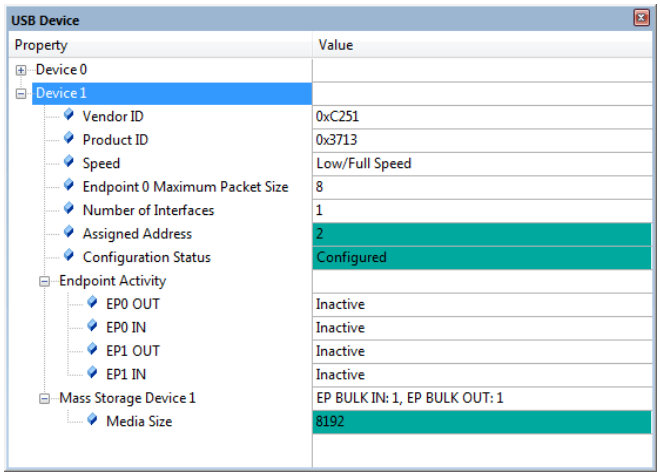
Once the project has its basic architecture, disable the use of latest versions of software packs in the Select Software Pack… dialogue box. A well-maintained project may not want changes to its source in an ad-hoc basis – source changes should be under a very deliberate control. Use of specific packs should be frozen…once you have the core of the project up and running.





Make use of custom Component View (.scvd) files

The µVision debugger has a convenient way to summarise a software component or system along with other debug information in its Component Viewer. Describing your system using a .scvd file allows a much more human-readable depiction of your running code. It is well-worth taking the time in the early phases of a project to get a useful representation. The files are an XML based description with plenty of text to get wrong at first.



In Summary

Not everything is as simple as flicking a switch and will need some setup and configuration but is well-worth doing. If you can get all of the above running, you're well on your way to an easier time for when things get tricky.

Happy coding!

Further Information

For more information visit our website: www.hitex.co.uk or get in touch: info@hitex.co.uk. You can also connect with us: LinkedIn