HOWTO - TI CCS scripting for profiling and module simulation

Nikolai Kim

University of Technology Vienna, Vienna, Austria,
Institute of Computer Languages,
WWW home page: `http://www.complang.tuwien.ac.at`

Keywords: Howto, TI CCS, profiling, simulation

1 TI Code Composer Studio

CCS is a collection of development tools that is available from Texas Instruments for developing, simulating, optimizing, debugging and deploying applications that target various CPU's and systems distributed by Texas Instruments. CCS features an IDE based on Eclipse, a compiler chain, a collection of emulation/simulation modules as well as plugins for interfacing the target hardware.

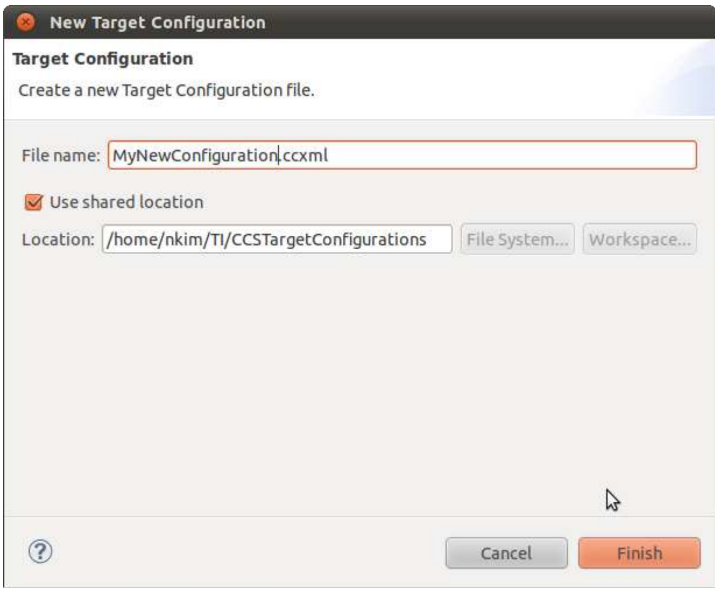

Fig. 1. Screen taking the name and location for a new target configuration.

Further, it provides a more or less flexible way of structuring the debugging/simulation process via API interfaces (there are basically two of them being of interest to us) accessible by *Java*, *JavaScript* and *GEL*. For the rest of this HOWTO we will concentrate on *JavaScript* only. Being colorful and functional, the IDE

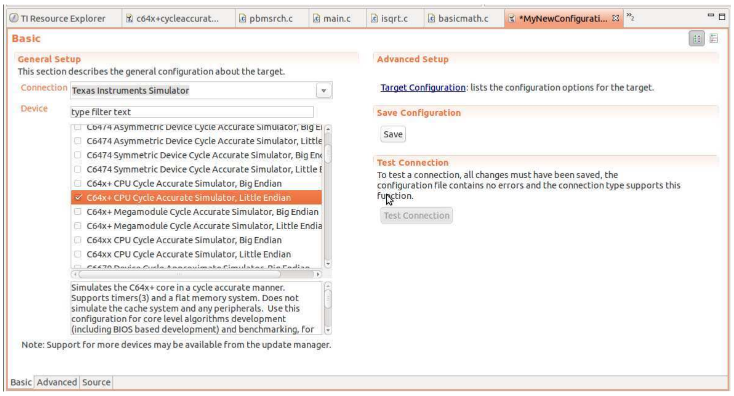

Fig. 2. Selection of a simulation model.

is still not really suitable for batch-processing due to the performance and accessibility issues. Thus, a console invokation is a better option for evaluating compiler performance. This can be done by adapting example files *loadti.sh* and *main.js* as shipped with the CCS and located (for CCS version 5.1.0.9000) in `<INSTALL-DIR>/ccsv5/ccs_base/scripting/examples/loadti`. Prior to simulator invokation, a target specification is required. CCS adapts the XML format for storing target configurations. Within IDE this can conveniently be done via *File-New-Target Configuration File*. Figure 1 shows the first screen which takes the name of a new configuration, figure 2 depics the screen where a simulation model can be specified. Here, we have specified a model for a C64x+ little endian CPU. After the configuration has been created, it can be saved (*Save* button). CCS stores this configuration in the location as specified by the first screen (Figure 1). For development within IDE, an additional step is required in order to build the target application properly. Figure 3 shows build options for a simple project. Needless to say, that settings done here must match the target configuration (especially endianness and the CPU model). If no intention of IDE development persists, this step can simply be skipped.

2 Cycle count and profiling

Counting cycles is not as trivial as one might think. This involves some scripting work invoking API-methods provided by the DebugServer interface. `http://processors.wiki.ti.com/index.php/Debug_Server_Scripting` describes basics of DSS scripting in detail, thus i skip it here. One major question is what

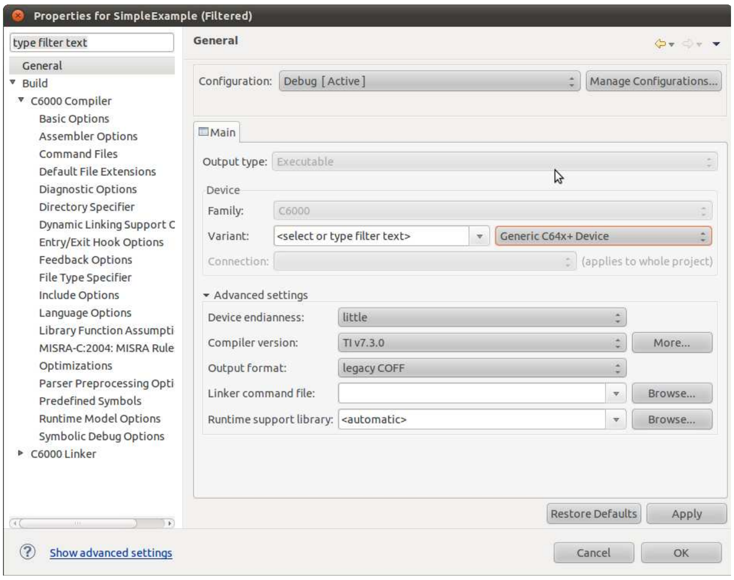

Fig. 3. Build options for a simple project.

cycles we are talking about. Pure CPU cycles can be obtained by the C64x+ cycle accurate simulation model, if however, total cycles are desired, *C64x+ MegaModule* simulation/target configuration must be used (Figure 2). Additionally, profiling (called *FunctionProfiling*) must be specified in order for the total counting to be functional. See `http://processors.wiki.ti.com/index.php/Profiling_with_DSS` for details. Having created a target configuration based on *C64x+ MegaModule* simulation, and given a profiling being set up correctly, we are still not able to generate reasonable numbers, since scripts shipped with CCSv5.1 do not implement reasonable cache size parameters. Therefore, the target machine description needs to be updated (*tisim_c64plus.cfg* located in `<INSTALL-DIR>/ccsv5/ccs_base/simulation/bin/configurations` for CCS v5.1).

Listing 1.1. Changing cache/memory parameters.

```
/////////////////////////////////////////////////////////
// PMC Configuration                                   //
//  *****************                                  //
//                                                     //
//  Total Mem @ Sim Init  :- 1   Mbytes ----(1)        //
//  Cache Size @ Sim Init :- 0   Kbytes ----(2)        //
// SRAM  Size @ Sim Init  :- (1) - (2) = 1   Mbytes    //
// SRAM Start Address     :- 0x00F00000                //
/////////////////////////////////////////////////////////
```

```
         MODULE DMC;
//            CACHE_SIZE              0x0000;
//            MEMORY_SIZE             0x100000;

              // NKim, 32k data cache
              CACHE_SIZE              0x8000;
              MEMORY_SIZE             0x8000;
              SRAM_START_ADDRESS      0x00F00000;
         END DMC;

 ////////////////////////////////////////////////////////
 // PMC Configuration                                  //
 // *****************                                  //
 //                                                    //
 // Total Mem @ Sim Init   :- 1    Mbytes ----(3)      //
 // Cache Size @ Sim Init  :- 0    Kbytes ----(4)      //
 // SRAM  Size @ Sim Init  :- (3) - (4) = 1  Mbytes    //
 // SRAM Start Address     :- 0x00E00000               //
 ////////////////////////////////////////////////////////
         MODULE PMC;
//          CACHE_SIZE              0x0000;
//          MEMORY_SIZE             0x100000;

            // NKim, 32k instruction cache
            CACHE_SIZE              0x8000;
            MEMORY_SIZE             0x8000;
            SRAM_START_ADDRESS      0x00E00000;
         END PMC;
```

Listing 1.1 shows changes for L1 program and data cache. For more detailed information, refer `http://processors.wiki.ti.com/index.php/C64x%2B_Cycle_Accurate_Simulator`.

After these steps we are done - and if no compatibility issues exist - can now produce reliable numbers for any (supported) profiling event of interest.

3 Compatibility

Linux CCS versions earlier than 5.1.0 have various bugs confirmed by TI developers. Profiling and therefore total cycle counting did not work for me in a reliable manner, due to missing/misplaced/broken *jar* files, especially for the *dvt scripting*. Version 5.1.0.09000 is the first stable release version in the v5 family. While working flawlessly on various Ubuntu clients (32/64 bits), Debian is currently not supported.

Thus, to make it useable on Debian machines some additional work may be required. Complang's *c8* machine hosts a Debian *lenny*, which ships a 32 bit version of *libstdc++.so.6* which is too old (GLIBCXX_3.4.9, while GLIBCXX_3.4.10

and newer is required for the CCS v5.1). This can easily be solved by providing a newer version of this lib. For me, building *gcc-4.4.5* and prepending *lib* and *lib32* in its installation directory to *LD_LIBRARY_PATH* did the trick (since libstdc++ is not shipped separately any longer). Not the most elegant way, but functional. If no drivers are required for hardware devices, the entire CCS setup can even be done having user privileges. See `http://processors.wiki.ti.com/index.php/Linux_Host_Support` for more details about Linux support. Also, consulting Texas Instruments developer forum (`http://e2e.ti.com/support/development_tools/code_composer_studio/f/81/t/3131.aspx`) may provide further insights, in case CCS increases your frustration level. Which for sure has the potential to do so.

4 Error messages

Loading device information failed right after the IDE startup most probably indicates obsolete libraries (see Section 3). If ignoring this and still enforcing the debug-machinery to run, you will run into **DebugServer XPCom registration** errors. Here the heartbeat stops, there is no way around this and you need to provide newer libraries. **DISPLAY not set** messages in turn indicate *absense* of 32-bit libraries required by CCS (even on 64 bit hosts). When running *JavaScript*'s you might encounter various runtime exceptions (*DVTServer, ProfilingAnalysis, etc.*) which indicate improper usage of supplied *jar* files and is a frequent case for CCS versions prior to the 5.1 release (these versions are not recommendable anyway due to the beta-nature and their limited distribution).

One additional thing needs to be considered. Memory and cache changes done to *tisim_c64plus.cfg* for the *C64x+ MegaModule* simulation need to be conform with the parameters specified by the linker file being in use. Ignoring this, you will run into **DMC/PMC alignment errors** during simulation for all but most trivial test cases.

5 EPICOpt LLVM compatibility

CCS internal profiling seems to rely on debug information produced and maintained properly by the compiler in use. According to TI manuals DWARF 2.0 is used for that purpose. While LLVM's debug information handling matures with each release (2.9 would most probably be robust enough for our basic experiments), there is no proper implementation of the debug information emission conform to the TI assembler yet. Thus, for now, there is unfortunately no simple solution to the overal cycle-/miss statistic generation for EPICOpt's LLVM-TI backend.