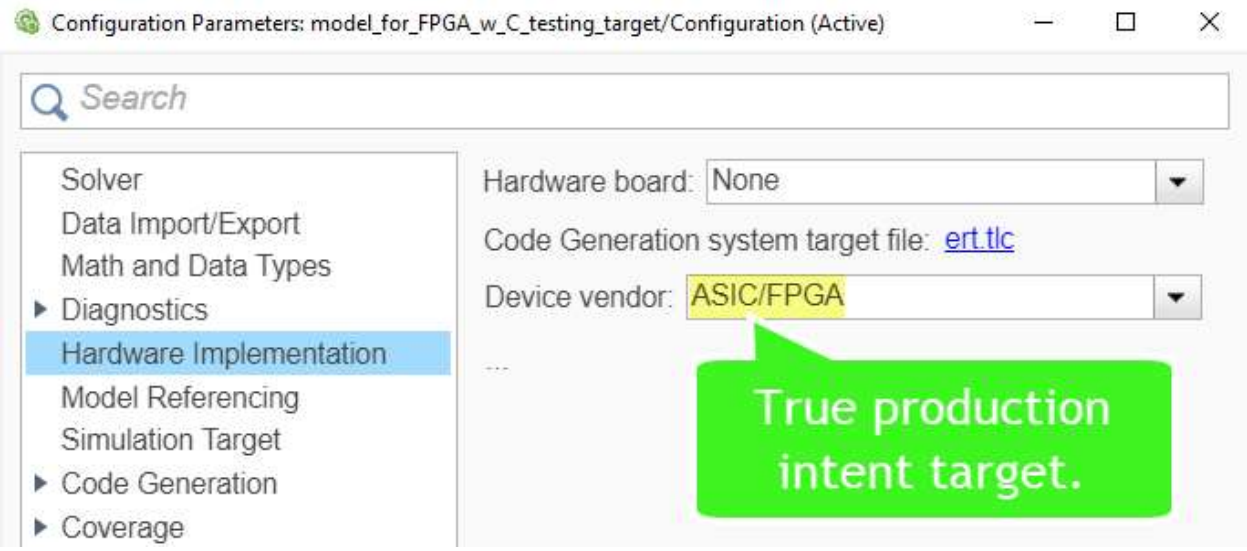
A Hidden Gem: Simulink's Coder Testing Target

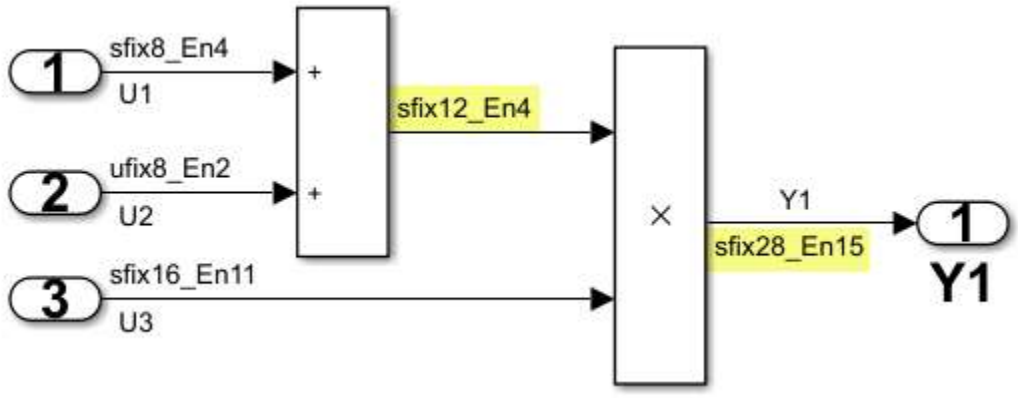
Andy Bartlett MathWorks 1/3/2020

Not too long ago I meet with about 20 Simulink users that all had a similar request. Their designs were going into production on an FPGA. But they wanted to simulate their designs using generated C on a different device. The problem was when they changed the target of their model it changed their design's behavior. They wanted to know how soon they could have the ability to keep their production target unchanged but be able to generate bit true C code for a different target. The good news is that this capability has already been shipping for about 2 decades as a "hidden gem". This article will explain this too well-hidden feature and show how to access it.

Consider an example model whose production intent target is an FPGA.

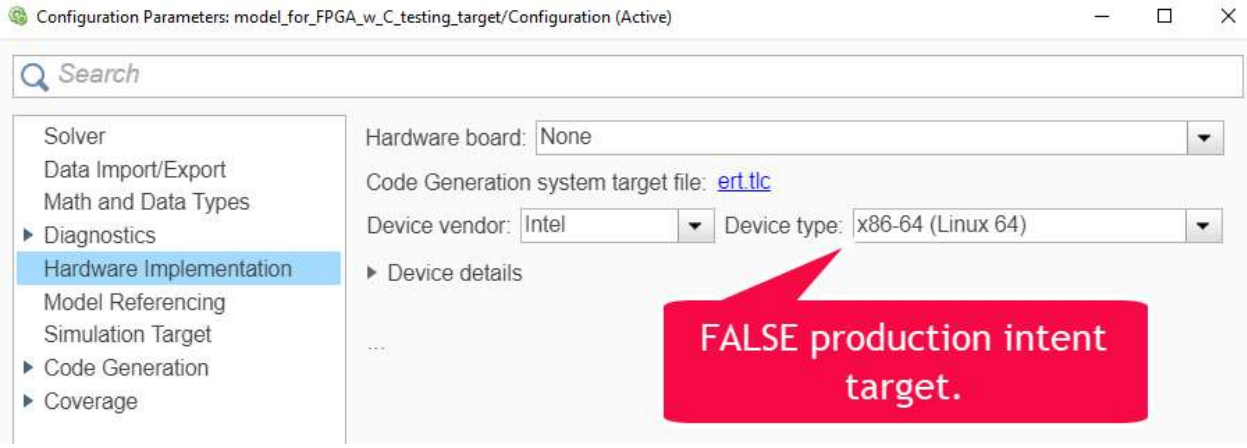


This model is using a variety of fixed-point type. Type propagation rules, knowing that ASIC/FPGA is the production intent, have automatically selected minimum word-length full-precision type for the output of two of the blocks. The custom word-lengths of 12 and 28 bits are sensible choices for an FPGA.

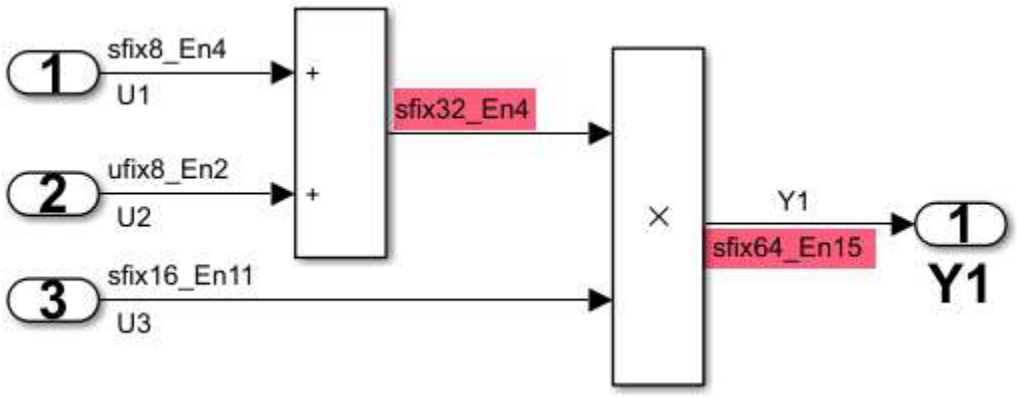


All is good, except the user wants to generate C code to simulate their design on a different microprocessor. They might want to simulate it on a cloud compute clusters using 64 bit x86 chips or

they do rapid-prototyping on embedded ARM chip. A FALSE approach to solving this need is to change the production intent target of the model

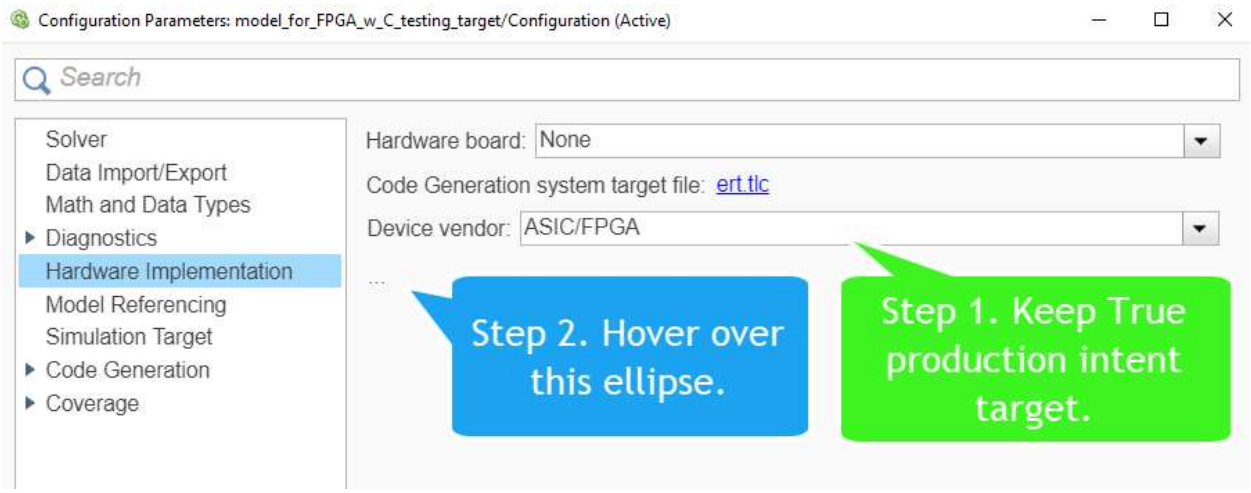


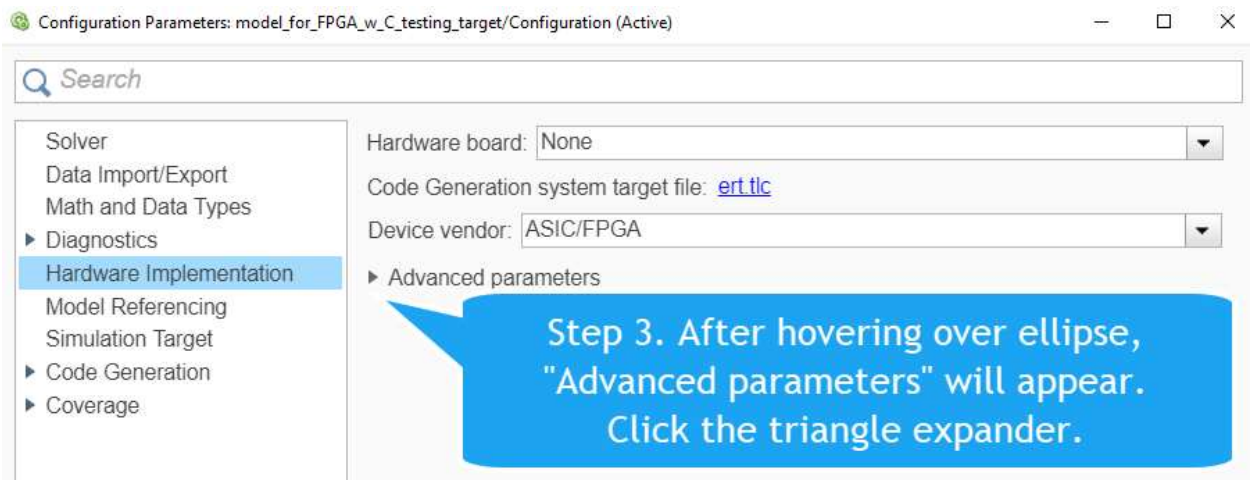
This false change to the production intent target will allow C code generation for a 64 bit Linux machine, but it has made undesired changes to the model.

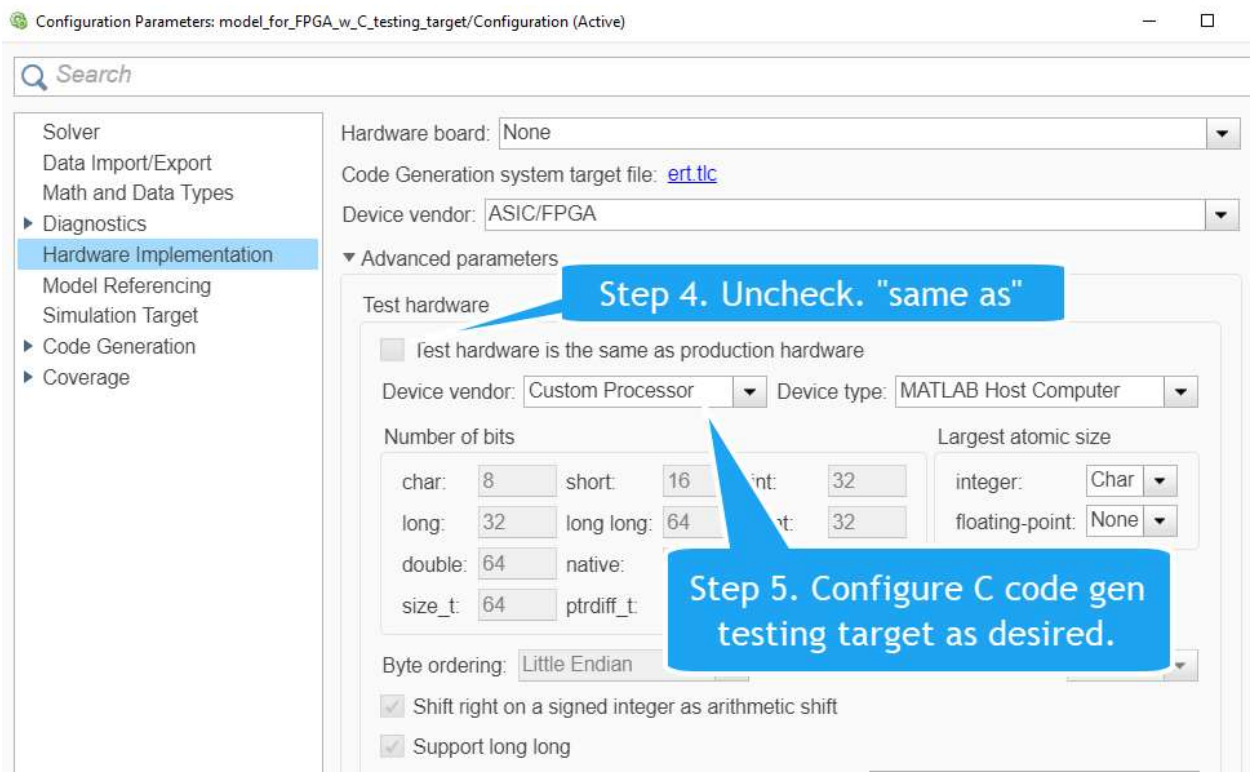


Notice that the sum and product block data types have changed from 12 and 28 bits to 32 and 64 bits, respectively. The data type propagation rules have now picked 32 and 64 bits because those are practical integer sizes for a x86 device. But the goal is to simulate the original design, not a modified design.

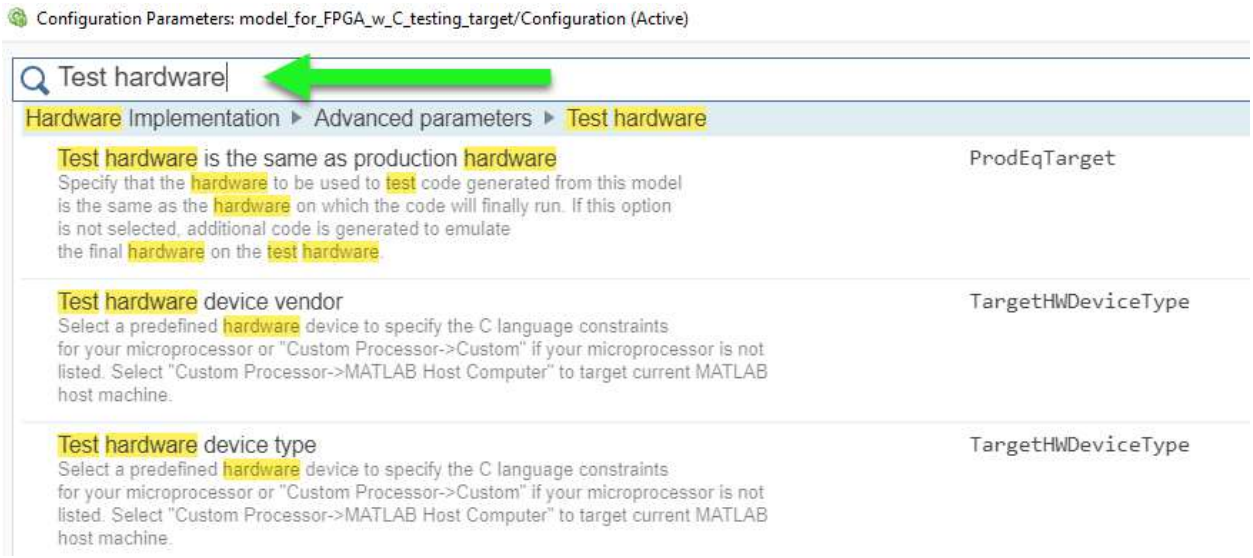
The win-win solution is to leave the production intent target unchanged and turn on and configure the testing target. The “secret password” to access this feature is to hover over the faint ellipse at the bottom of Hardware Implementation dialog. When the triangle shaped expander labelled “Advance Parameters” appears click it to show testing target controls. To enable a different C code generation target from the production intent one, uncheck “Test hardware is the same as production hardware.” Once uncheck, all the choices for the testing target will appear including Device vendor, bits per long, etc.


Configuration Parameters: model_for_FPGA_w_C_testing_target/Configuration (Active)
Search
Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
Hardware board: None
Code Generation system target file: ert.tlc
Device vendor: ASIC/FPGA
...
Step 2. Hover over this ellipse.
Step 1. Keep True production intent target.


Configuration Parameters: model_for_FPGA_w_C_testing_target/Configuration (Active)
Search
Solver
Data Import/Export
Math and Data Types
Diagnostics
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
Hardware board: None
Code Generation system target file: ert.tlc
Device vendor: ASIC/FPGA
Advanced parameters
Step 3. After hovering over ellipse, "Advanced parameters" will appear. Click the triangle expander.



An alternate way to access these controls is to type “Test hardware” in the Configuration Parameters search box. Then, click on the first choice “ProdEqTarget” shown.



Having the testing target turned on only effects the generated C code. It has no effect type propagation, simulation, analysis, etc. The C code that is generated for the testing target will be as numerically faithful to the production intent design as possible. For integer and fixed-point math, the agreement will be bit-true. For floating-point maximum agreement is a goal, but some small numeric differences may exist.

If we inspect the C code that was generated for the testing target, we see a lot of bitwise ANDs and ORs with values like 2048 and 134217728. These funny looking operations are forcing the types available on

the testing target to emulate the 12 and 28 bit types that will be used on the production intent FPGA. These details are what gives the testing target code bit-true agreement with the production integer and fixed-point math.

```
48      tmp = ((tmp_0 & 2048U) != 0U ? tmp_0 | -2048 : tmp_0 & 2047) * U3;
49      Y1 = (tmp & 134217728U) != 0U ? tmp | -134217728 : tmp & 134217727;
```

The motivating use case discussed here was FPGA for production. But the feature works equally well when the production target is a microprocessor and the testing target is a different microprocessor. For example, the production target could be an low-power 16-bit PIC microcontroller and the testing target could be a DSP based rapid prototyping system or a 64 bit server class ARM in a compute cloud. You have full mix-and-match capability. You just must know how to find this hidden gem.

Thank you for reading

Andy Bartlett

PS: If you'd like to play with this example, you can get the examples shown on GitHub.

https://github.com/mathworks/NumericEfficiencyExamples

The example models and library block will be found in the folder named HiddenGemCodeGenTestingTarget.

Models are provided that can be used in release R2015a or newer.