How to Dramatically Increase Efficiency of Simulink Generated Code in Two Clicks!

Andy Bartlett MathWorks 1/2/2020

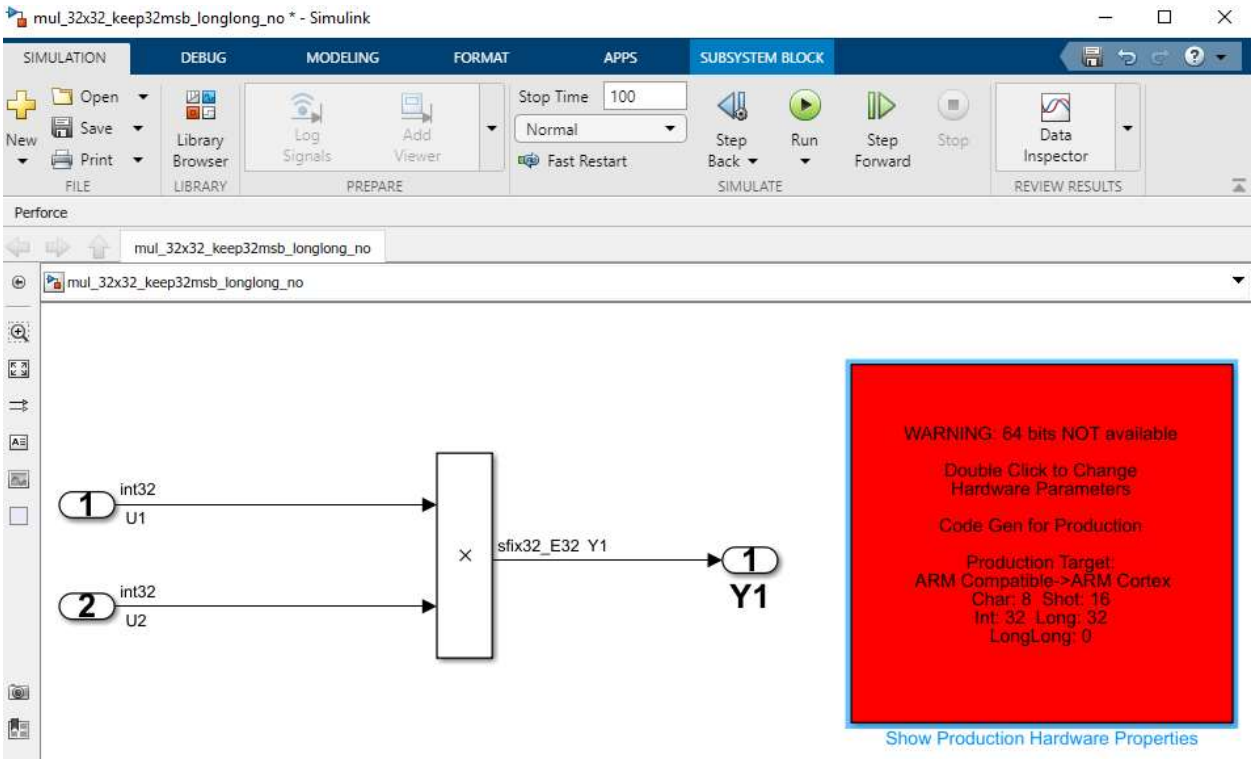
This article will show you how to make C code generated from MATLAB and Simulink dramatically more efficient with two simple clicks! A recent user question shows that it is easy to be unaware of this powerful capability. This article will show a modeling utility that makes it easy to see the cause of inefficiency and how to correct it.

The user wanted code like the following 60+ lines to be more efficient.

```
void mul_wide_s32(int32_T in0, int32_T in1, uint32_T
*ptrOutBitsHi, uint32_T
                  *ptrOutBitsLo)
{
  uint32_T absIn0;
  uint32_T absIn1;
  uint32_T in0Lo;
  uint32_T in0Hi;
  uint32_T in1Hi;
  uint32_T productHiLo;
  uint32_T productLoHi;
  absIn0 = in0 < 0 ? ~(uint32_T)in0 + 1U : (uint32_T)in0;
  absIn1 = in1 < 0 ? ~(uint32_T)in1 + 1U : (uint32_T)in1;
  in0Hi = absIn0 >> 16U;
  in0Lo = absIn0 & 65535U;
  in1Hi = absIn1 >> 16U;
  absIn0 = absIn1 & 65535U;
  productHiLo = in0Hi * absIn0;
  productLoHi = in0Lo * in1Hi;
  absIn0 *= in0Lo;
  absIn1 = 0U;
  in0Lo = (productLoHi << 16U) + absIn0;
  if (in0Lo < absIn0) {
    absIn1 = 1U;
  }

  absIn0 = in0Lo;
  in0Lo += productHiLo << 16U;
  if (in0Lo < absIn0) {
    absIn1++;
  }

  absIn0 = (((productLoHi >> 16U) + (productHiLo >> 16U)) +
in0Hi * in1Hi) +
    absIn1;
  if ((in0 != 0) && ((in1 != 0) && ((in0 > 0) != (in1 > 0)))) {
    absIn0 = ~absIn0;
    in0Lo = ~in0Lo;
```

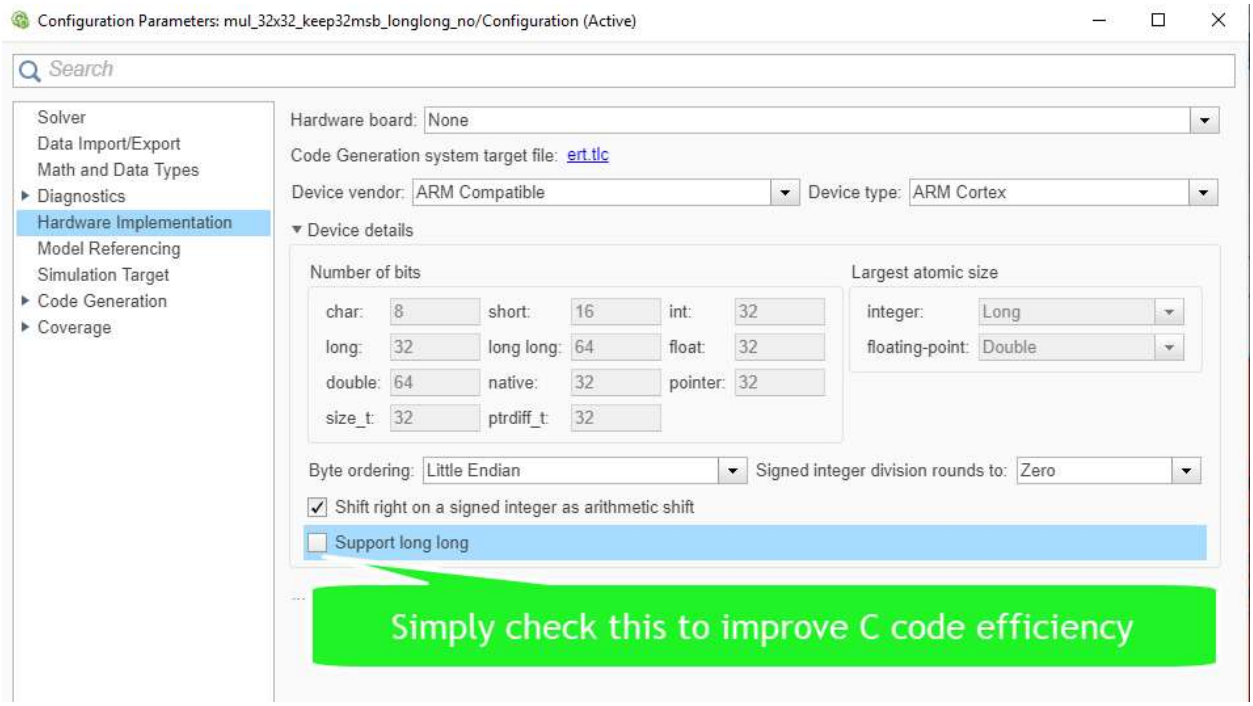
```
    in0Lo++;
    if (in0Lo == 0U) {
      absIn0++;
    }
  }

  *ptrOutBitsHi = absIn0;
  *ptrOutBitsLo = in0Lo;
}

int32_T mul_s32_sr32(int32_T a, int32_T b)
{
  uint32_T u32_chi;
  uint32_T u32_clo;
  mul_wide_s32(a, b, &u32_chi, &u32_clo);
  return (int32_T)u32_chi;
}

/* Model step function */
void mul_32x32_keep32msb_longlong_no_step(void)
{
  Y1 = mul_s32_sr32(U1, U2);
}
```

By dropping a utility block on the model, the missed opportunity becomes readily apparent.

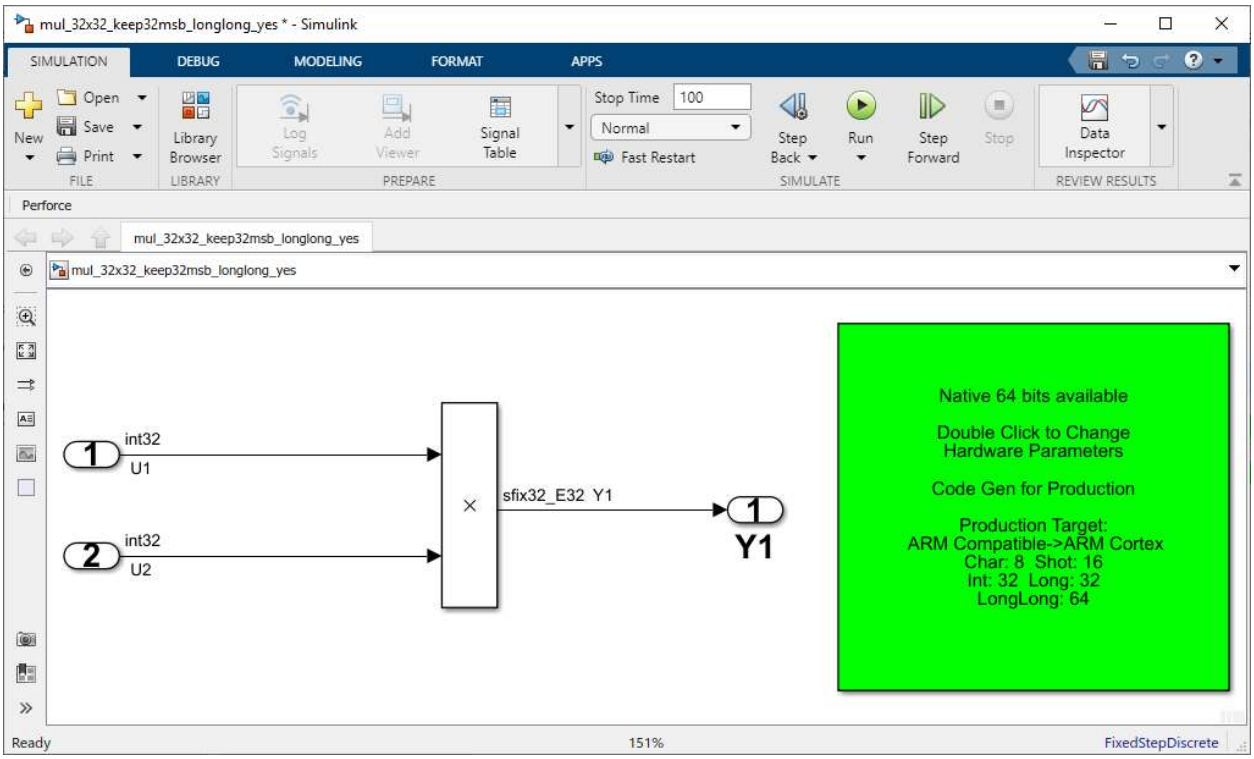


The code generation target is configured to use an ARM Cortex, but it has **not** be configured to make the 64 bit long long data type available. The long long data type officially became part of the C standard way back in 1999. The long long data type has been supported by Embedded Code and by ARM Compilers for many years. There does not seem to be any good reason not to exploit compiler supported access to 64 bit integer types. (If you are stuck to 30 year old 1989 C standard or have another reason you cannot use long long, please send me a message and let me know your use case.)

To leverage long long, simply click on the red block, to open the Configuration Parameters dialog to the critical parameter.



Once support for long long is enabled, the utility block will turn a happy green.



More importantly, if Embedded Coder is used to regenerate code, it will be dramatically more efficient. In fact, it will be comprised of just one simple expression.

```
/* Model step function */
void mul_32x32_keep32msb_longlong_yes_step(void)
{
  Y1 = (int32_T)(((int64_T)U1 * U2) >> 32);
}
```

It doesn't get much easier than that. With literally two clicks, the long long data type was enabled for C code generation from the Simulink model. For math operations that require 64 bit integer or fixed-point math, utilizing long long can dramatically improve efficiency. For the example shown, the improvement was roughly 60X. Not bad for two clicks.

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 MoreEfficientCodeInTwoClicks.

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