

Latest Features in Simulink Fixed Point

March 2010

R2010a

Demo Download

The demos presented in this document can be found in the shipping version of the product or at www.mathworks.com/support/solutions/en/data/1-BWO86Z.

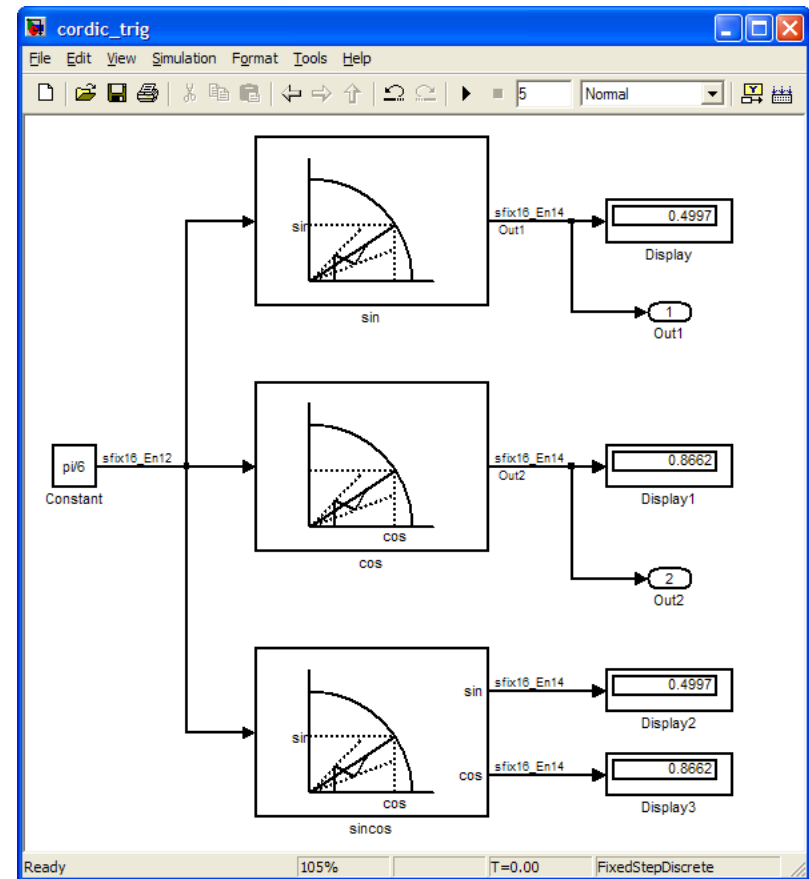
CORDIC Approximation in Trigonometric Function Block

Use CORDIC approximations in trigonometric functions to save memory over lookup table approaches

- Supports sin, cos, and sincos functions
- Supports fixed-point simulation
- Supports C and HDL code generation

```

xn = 9949;
yn = 0;
for (iter = 0; iter < 11; iter++) {
    y = (yn >> iter);
    x = (xn >> iter);
    if (angle < 0) {
        xn += y;
        yn -= x;
        zn += cordic01_P.TrigFunction_lutData[iter];
    } else {
        xn -= y;
        yn += x;
        zn -= cordic01_P.TrigFunction_lutData[iter];
    }
}
    
```

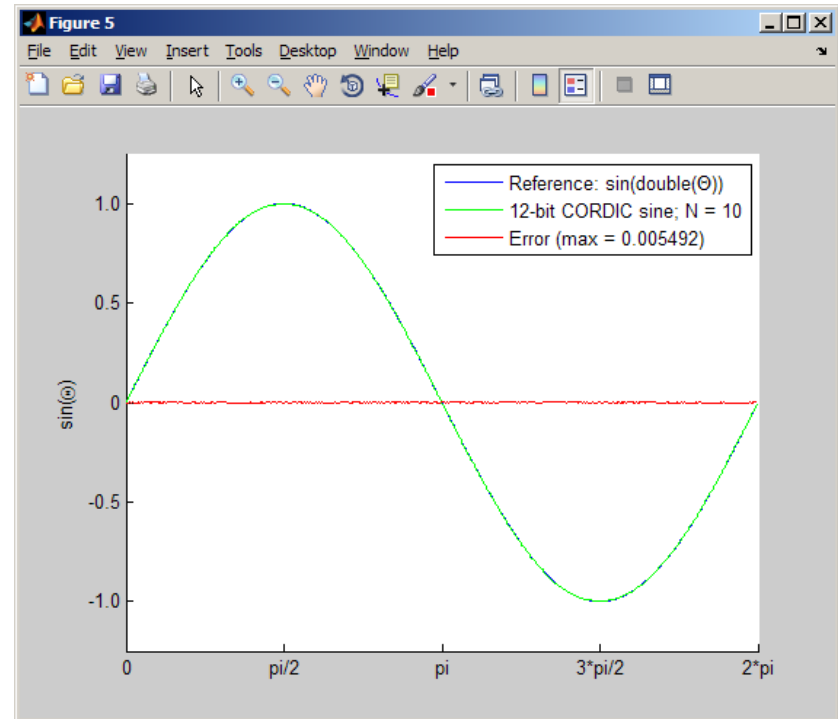


Demo: >> cordic_trig

CORDIC Approximations in Embedded MATLAB Functions

Use CORDIC approximations in trigonometric functions to save memory over lookup table approaches

- Supports `sin`, `cos`, and `sincos` functions
- Supports fixed-point simulation
- Supports C and HDL code generation



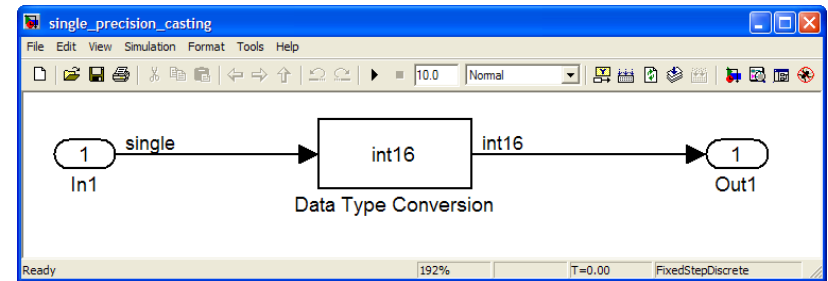
12-bit CORDIC sine
CORDIC algorithm iterations = 10

Demo: `>> cordic_example`

Improved Casting in Algorithms Using Single-Precision and Fixed-Point Data

Use direct casts from single-precision to fixed-point data in algorithms that use both data types

- Reduces RAM and ROM
- Improves execution speed



```
void single_precision_casting_step(void) R2009b
{
    real T tmp;
    real T tmp_0;
    tmp = (real T)U.In1;
    tmp_0 = (real T)(real32 T)tmp;
    if ((tmp_0 < 4.5035996273704960E+015) && (tmp_0 > -
4.5035996273704960E+015)) {
        tmp_0 = floor(tmp_0 + 0.5);
    }
    if (rtIsNaN(tmp) || rtIsInf(tmp)) {
        tmp = 0.0;
    } else {
        tmp = fmod(tmp_0, 65536.0);
    }
    Y.Out1 = tmp < 0.0 ? (int16 T)-((int16 T)(uint16 T)(-tmp)) : (int16 T)
(uint16 T)tmp;
}
```

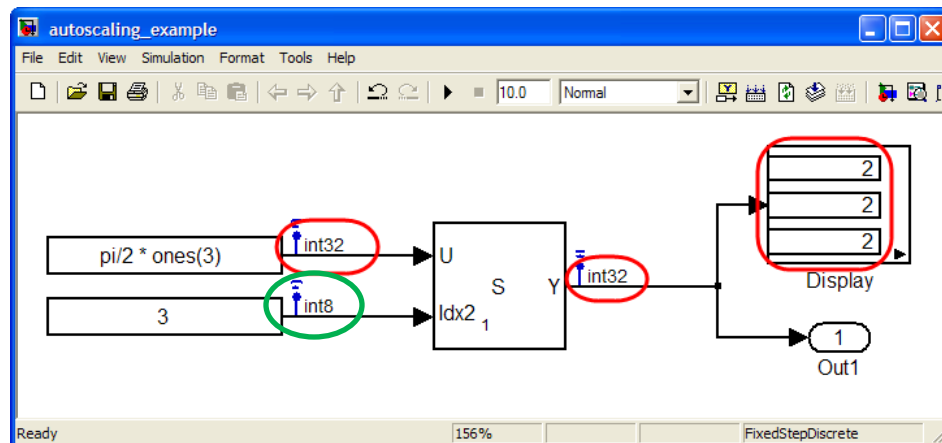
```
void single_precision_casting_step(void) R2010a
{
    real32 T tmp;
    tmp = U.In1;
    if ((U.In1 < 8.388608E+006F) && (U.In1 > -8.388608E+006F)) {
        tmp = (real32 T)floor(U.In1 + 0.5F);
    }
    if (rtIsNaNF(U.In1) || rtIsInfF(U.In1)) {
        tmp = 0.0F;
    } else {
        tmp = (real32 T)fmod(tmp, 65536.0F);
    }
    Y.Out1 = (int16 T)(tmp < 0.0F ? (int32 T)(int16 T)-
((int16 T)(uint16 T)(-tmp))) : (int32 T)(int16 T)(uint16 T)tmp);
}
```

Demo: >> single_precision_casting

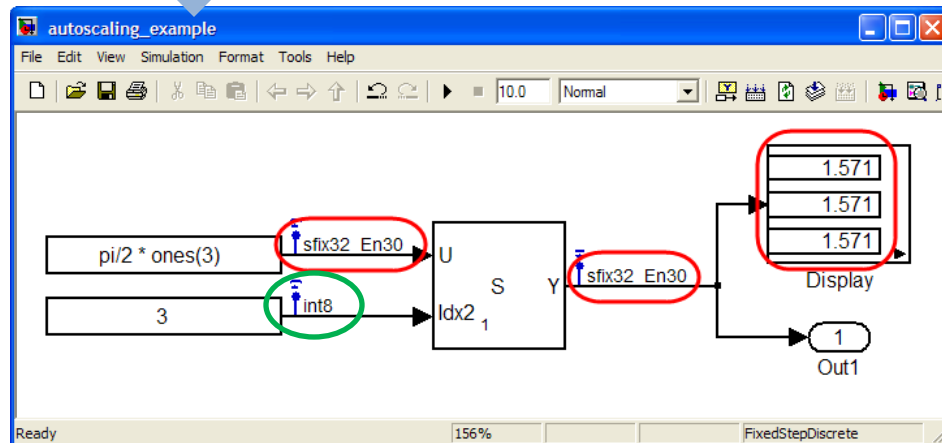
Automatic Scaling Improvements

Smoother workflow when accepting autoscale proposals

- Better handling of data type constraints for several Simulink blocks using autoscaling in Fixed-Point Tool and Fixed-Point Advisor
- Better overflow handling and improved precision
- Detection of potential numeric problems in Fixed-Point Tool



Autoscale

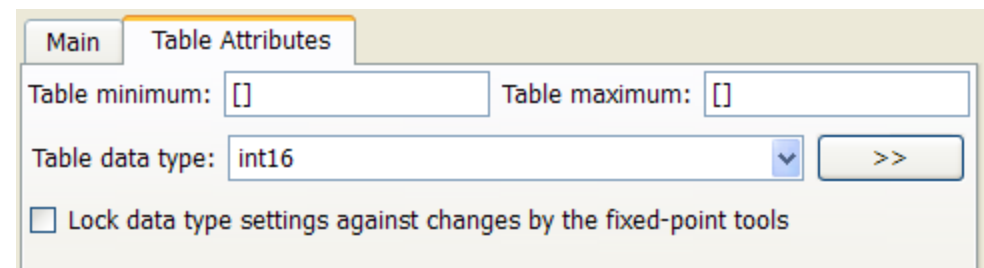
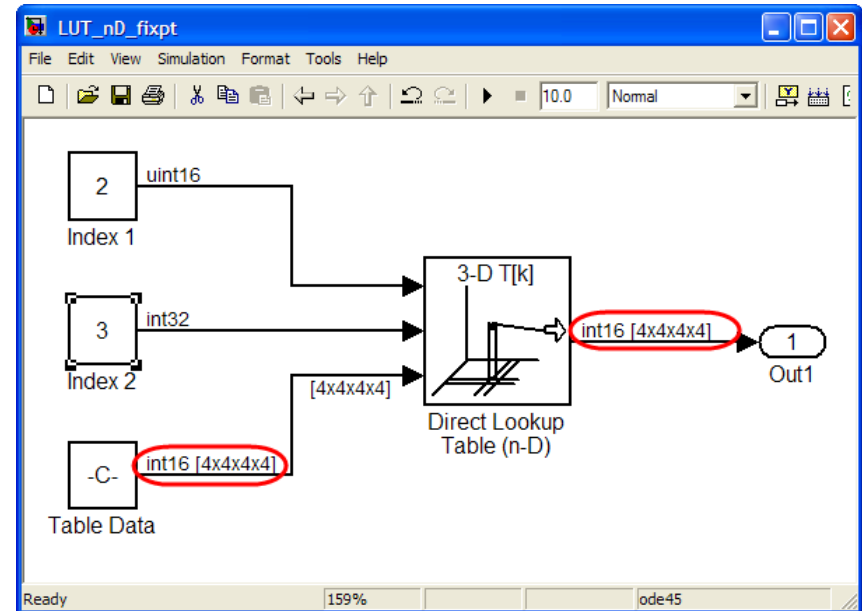
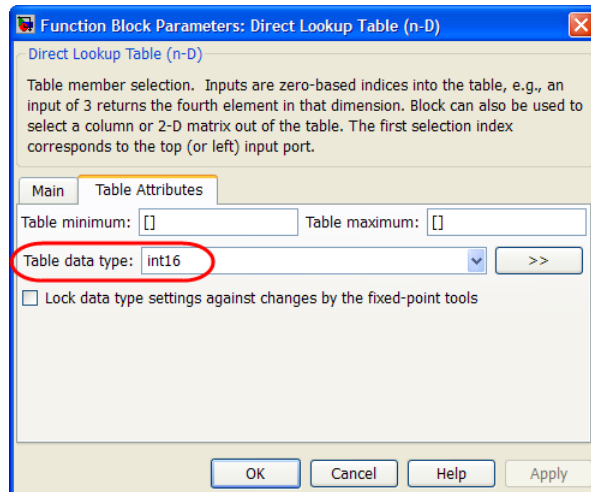


Demo: >> autoscaling_example

Direct Lookup Table (n-D) Block Enhancements

Use fixed-point data types in Direct Lookup Table (n-D) blocks

- Ability to enter table via input port or fixed-point table data
- Support for n-dimensional table data when entering via input port



Demo: >> LUT_nD_fixpt

Improvement in Division by Constant Power of 2

Generate more efficient code when using signed division by constant power of 2 with simplest rounding

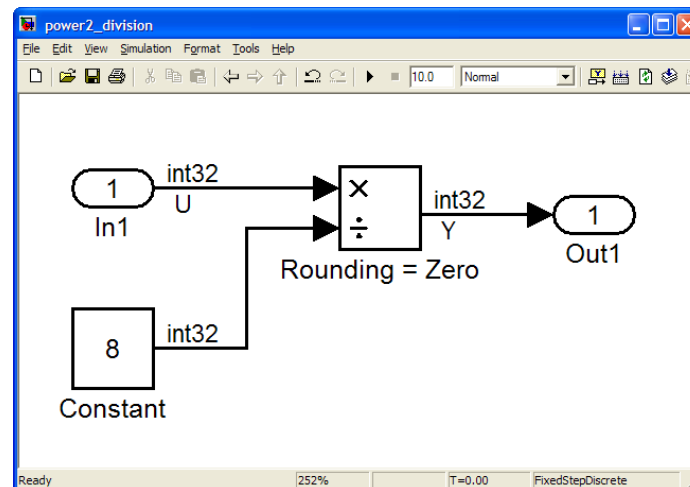
Users can:

- Use regular division that naturally rounds to zero
- Potentially use less code

Target compiler can:

- Optimize code in a more target-relevant manner
- Combine the plain division with other optimizations

Demo: `>> power2_division`



Big cast with rounding to zero

```
Y = ((U < 0) && ((U & 7) != 0)) + (U >> 3);
```

R2009b

Plain division – rounds to zero naturally
Compiler will optimize

```
Y = U / 8;
```

R2010a