



# VisSim Training

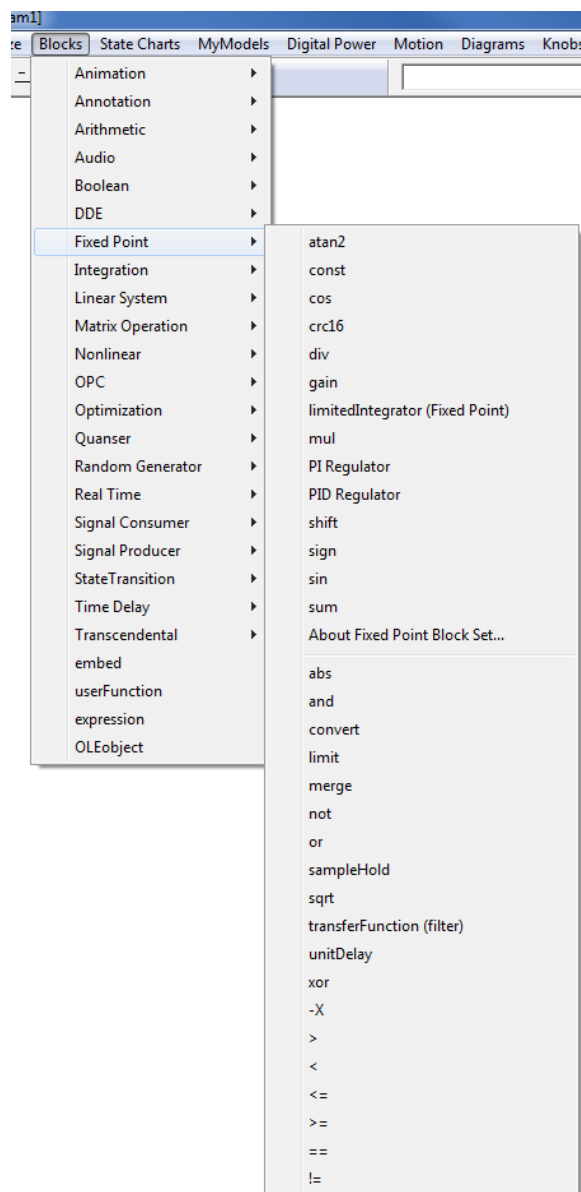
## 7. Fixed Point Arithmetic & Filters

## Topics:

---

- Fixed Point BlockSet
- Fixed Point Fundamentals Video
- Configuring a Fixed Point Block, “const” example
- Displaying Overflow Messages
- Using the Autoscale Feature
- Autoscale Video
- Code Generation, commenting, in line functions
- %CPU Utilization Example – discrete filter
- Discrete fixed point filter %CPU Comparison Video

# Fixed Point Blockset



The 33 element VisSim EMBEDDED fixed point blockset (“Blocks/ Fixed Point”) is used to design and simulate performance of fixed point algorithms prior to codegen and execution on an embedded platform.

## Fixed Point Block Features:

- Automatic radix point scaling
- Overflow alerts
- High & Low levels to determine optimal radix point settings
- Master control for all fixed point blocks

## Fixed Point Code Generation Features:

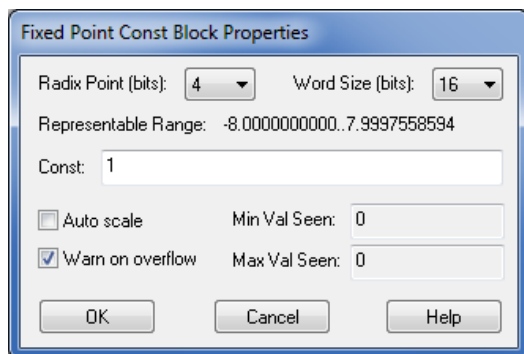
- Highly efficient code using in-line shifts
- Automatic commenting to enhance readability

[Fixed Point Menu](#)

[Fixed Point Fundamentals Video](#)

## Fixed Point Configuration - “const” block

1@fx4.16



Radix Point (bits): Analogous to the decimal point in a base 10 number.

Word Size (bits): Total number of bits in the fixed point number, set equal to the wordsize for the Target architecture.

Example:  $4.16 =$  

Maximum Value =  $0111.1111111111111111 = 7.9997558594$

Minimum Value =  $1000.0000000000000000 = -8$

Representable Range: -8 to 7.9997558594

Const: Entered in floating point representation.

**Precision**: Smallest step (difference) between two consecutive N bit number values

Example: 4.16: precision =  $2^{-12}$

Example: 1.16: precision =  $2^{-15}$

Fixed point targets only recognize integer values. VisSim EMBEDDED codegen automatically converts decimal numbers to scaled integer values based on the Radix Point and Wordsize settings. Comments, indicating the original Const value, are added to VisSim EMBEDDED codegen on each conversion

Auto scale: Resets the “Representable Range” when the maximum or minimum values are exceeded.

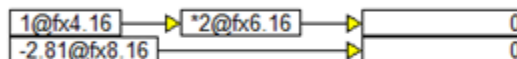
Warn on overflow: Presents a dialog box indicating an overflow (used in conjunction with “Fixed Point Block Set Configure...”)


Min Val Seen & Max Val Seen: watermarks of minimum and maximum values passed through the block

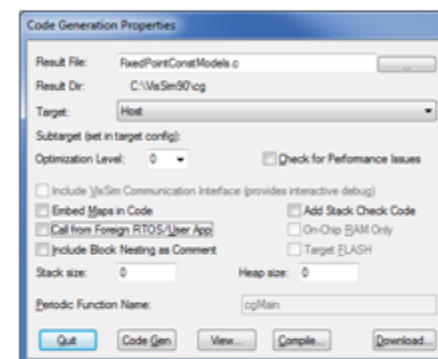
## Code Generation – Commenting, Inline Shifts

To view the automatic commenting and efficient execution features of VisSim EMBEDDED fixed point code generation, we will illustrate code gen for a simple model using a “Host” Target.

1. VisSim model consisting of “const” and “gain” blocks (“Blocks/Fixed Point”), and “display” (“Blocks/Signal Consumers”) blocks.



2. Configure “Code Generation Properties” under (“Tools/ Code Gen...”) as shown to the right. Click “Code Gen”, then “View...” 



3. The Code Gen "c" file will appear in "notepad" ➡

Scaled Integer:

-2.81@fx8.16=

$$-719 = (-2.81 / (2^{-8}) = -719)$$

- Inline shifts (multiply & divide) of scaled integers for efficient execution.

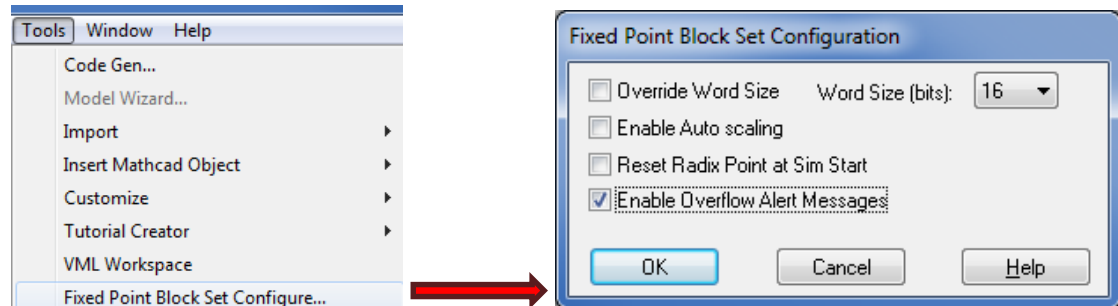
Comment of original floating point value

## Code Generation Example

## Displaying Overflow Messages

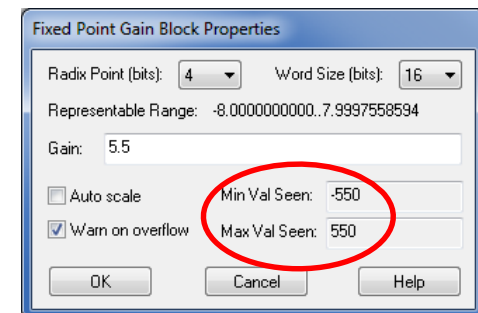
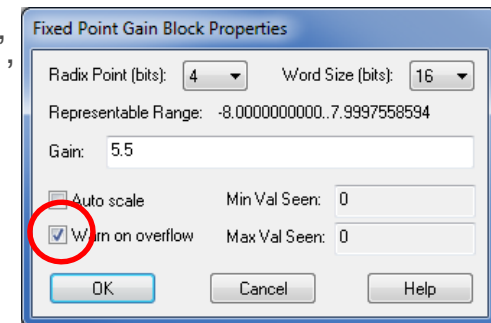
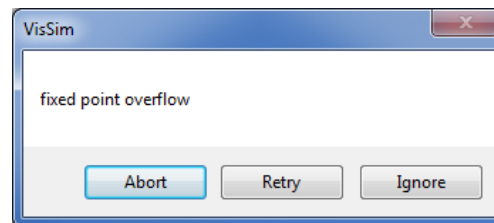
Overflow: The situation where the minimum or maximum value exceeds the “Representable Range”. Use the following procedure to observe overflow in a fixed point “gain” block.

1. In the “Fixed Point Block Set Configuration” (“Tools/Fixed Point Block Set Configure...”), check the “Enable Overflow Alert Messages”



2. In the fixed point “gain” block, “Fixed Point Gain Block Properties”, check the “Warn on overflow”

3. Apply a “slider” input = +/-100, Click “Go” to run the simulation, an overflow is detected and the following message will be displayed:

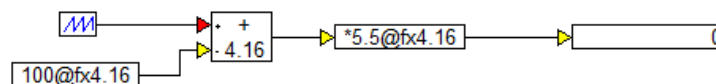


In the “Fixed Point Gain Block Properties”, the Min Val Seen and Max Val Seen display the high and low water marks of values that have passed through the block.

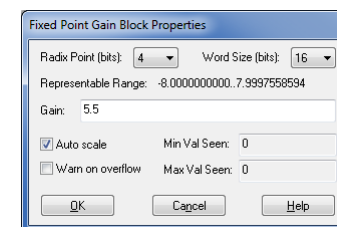
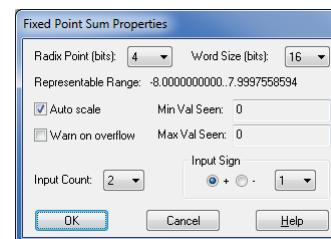
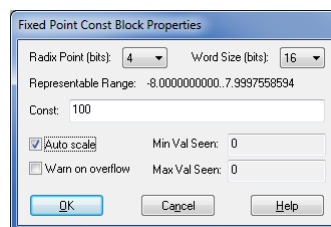
## Using the Autoscaling Feature

For each fixed point block, VisSim EMBEDDED maximizes precision by selecting the smallest acceptable range for each fixed point block in a model.

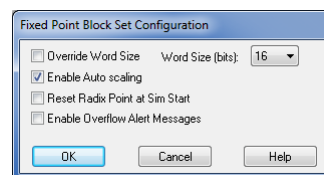
1. A model is created consisting of a:  
“sawtooth” with “Amplitude” = 200 from  
    (“Blocks/Signal Producer”)  
“const” from (Blocks/Fixed Point)  
“add” from (“Blocks/ Fixed Point”)  
“gain” from (“Blocks/ Fixed Point”)



2. Each fixed point block is configured as shown to the right:



3. In the “Fixed Point Block Set Configuration” (“Tools/Fixed Point Block Set Configure...”), check the “Enable Auto Scaling” option.

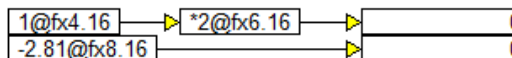


4. Click “Go” to run the simulation, the fixed point blocks being autoscaled whose output value exceeds the Min Val Seen or Max Val Seen will turn red, at the end of the simulation, each fixed point block will display the modified radix point value that provides an acceptable Representable Range for the simulation signals.

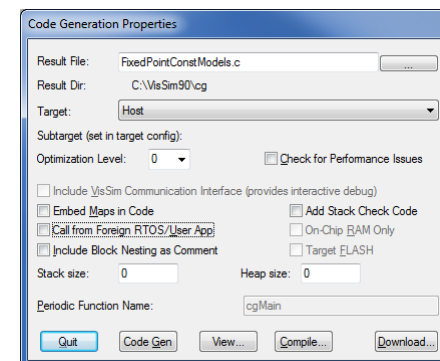
## Code Generation – Commenting, Inline Shifts

To view the automatic commenting and efficient execution features of VisSim EMBEDDED fixed point code generation, we will illustrate code gen for a simple model using a “Host” Target.

1. VisSim model consisting of “const” and “gain” blocks (“Blocks/Fixed Point”), and “display” (“Blocks/ Signal Consumers”) blocks.



2. Configure “Code Generation Properties” under (“Tools/ Code Gen...”) as shown to the right. Click “Code Gen”, then “View...” 



3. The Code Gen "c" file will appear in "notepad" →

The screenshot shows a Notepad window titled "FixedPointConstModels.c - Notepad". The menu bar includes File, Edit, Format, View, and Help. The text area contains the following C code:

```

/** Vissim Automatic C Code Generator Version
9.0A21 ***/
/* Output for FixedPointConstModels.vsm at Tue
Sep 01 16:34:39 2015 */

#include "math.h"
#include "cgen.h"

extern CGDOUBLE Zed;

static void cgMain(void);
static SIM_STATE tSim={0, 0.01,
10,0,0.01,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,cgMain,0,0,0,0,0,0,0},
SIM_STATE *sim=&tSim;
static void cgMain()
{
    int t5;
    int t3;
    t5 = MUL_SHIFT16(4096 * 1@fx4.16 * /,16384,15)/
2@fx6.16 -719 -2.81@fx8.16 */;
    display(( t3 * 0.003963625));
    display(( t3 * 0.00390625));
}

int main()
{
    simInit(&tSim);|
    startSim();
    return 0;
}

```

Red circles are drawn around the values 4096, 1@fx4.16, 16384, 15, 2@fx6.16, -719, and -2.81@fx8.16 in the code.

- Inline shifts (multiply & divide) of scaled integers for efficient execution.

### Scaled Integer:

-2.81 @fx8.16 =

$$-719 = (-2.81/(2^{-8}) = -719)$$

Comment of original floating point value



## Fixed Point Arithmetic – CPU Utilization Example

This example illustrates the CPU time savings using fixed point arithmetic instead of floating point to implement a digital filter.

The digital filter transfer function is:  $9.82585231224611e-006 \frac{z^2 + 2.000009738736z + .99997060955299}{z^2 - 1.9911045826979z + .99114388591402}$

The digital update time is: 0.001 seconds.

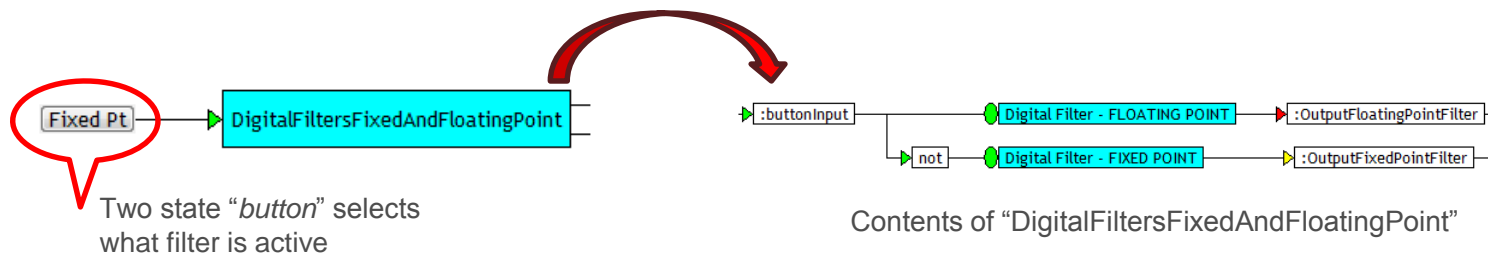
Two versions of the digital filter transfer functions are implemented,

Digital Filter – FLOATING POINT

Digital Filter – FIXED POINT

The input to each filter is attached to analog input 0 which is pinned out on the F28069M LaunchPad board. By placing your finger between J1 and J3 pins on the LaunchPad, an analog input signal is created.

The top level VisSim model has the two digital filters in the compound block “DigitalFiltersFixedAndFloatingPoint”

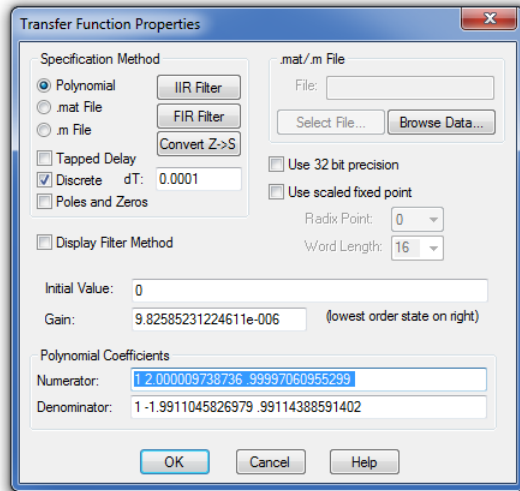
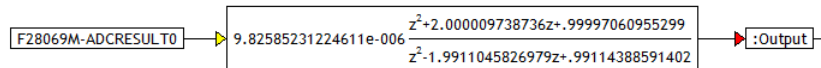


Contents of “DigitalFiltersFixedAndFloatingPoint”

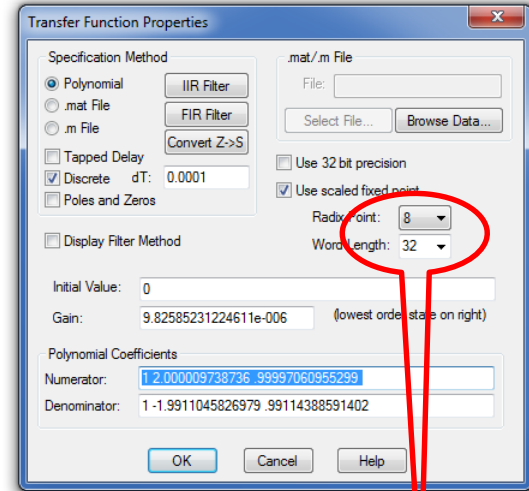
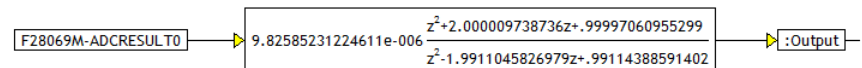
Note: “buttonInput” selects either filter

# CPU Utilization Example - Filter Configurations

## Digital Filter – FLOATING POINT Transfer Function Properties



## Digital Filter – FIXED POINT Transfer Function Properties



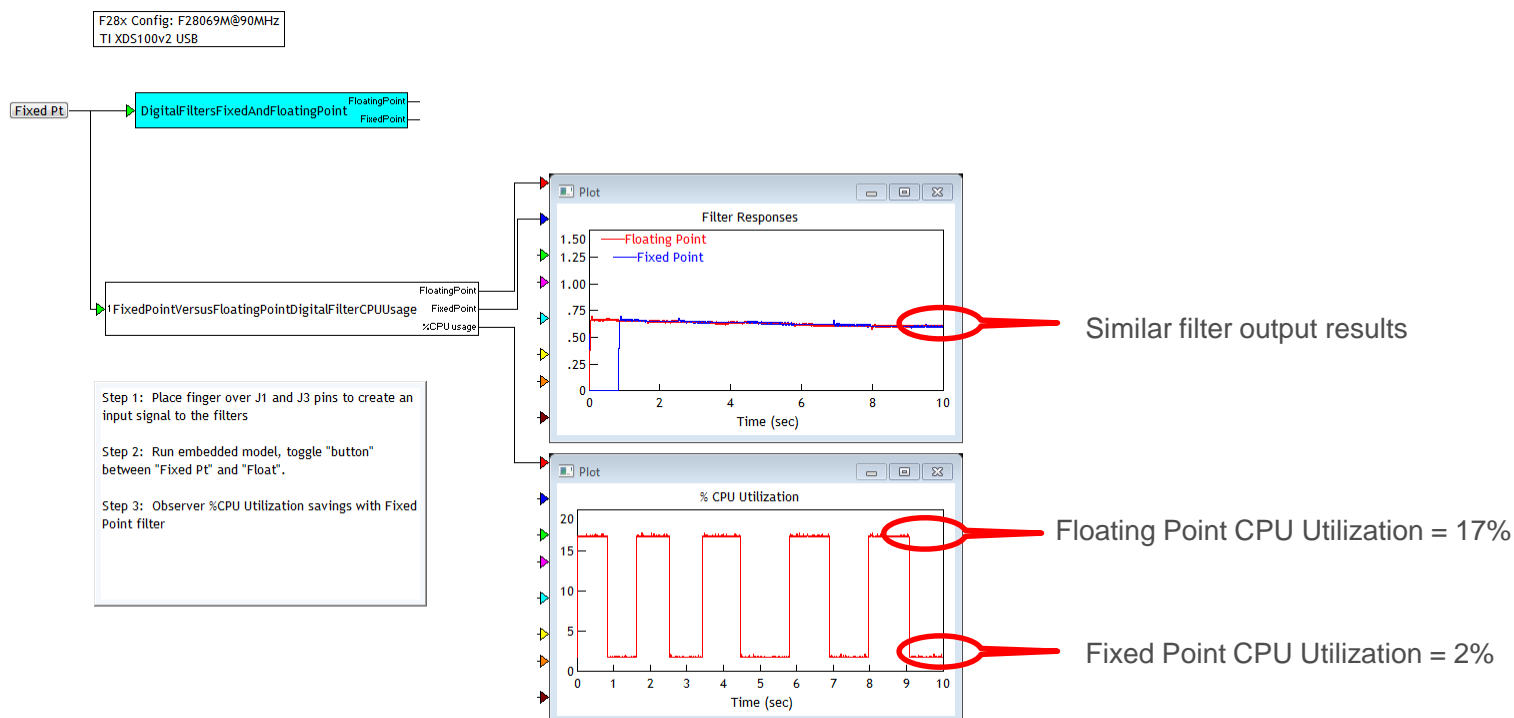
Fixed Point Format Selected 8.32

[View source model in VisSim](#)

[Fixed Point Filter Video](#)

## CPU Utilization Example - Results

Code is generated for the “DigitalFiltersFixedAndFloatingPoint” compound block and executed in the “target Interface” (below).



NOTE: Up until now, we have developed separate "Source" and "Debug" models. When the JTAG communication transfer speed can be sufficient (for the model being used) it is possible to combine the "Source" and "Debug" models into one "Source" model which includes the "target Interface".

## Summary

---

- When a target does not support hardware multiply or divide, the operations must be performed in software.
  - A software divide is approximately 100x slower than a hardware multiply, add, or shift
  - A software square root involves several divide iterations and is approximately 200x to 300x slower than a hardware multiply, add, or shift
- Using Fixed Point arithmetic greatly reduces the CPU Utilization required for software multiplies, divides, and other complex operations.
  - In the digital filter example, the fixed point implementation used 1.8% CPU while the floating point implementation used 16.76% CPU, almost a factor of 10x savings in %CPU Utilization.
- VisSim EMBEDDED Fixed Point blockset provides an easy and efficient way to migrate control algorithms to fixed point implementations that meet target hardware limitations and CPU Utilization requirements.



## Thank You

For more information please visit;

<http://www.altairhyperworks.com>

Search for VisSim under the Products category