Taighde Éireann
Research Ireland

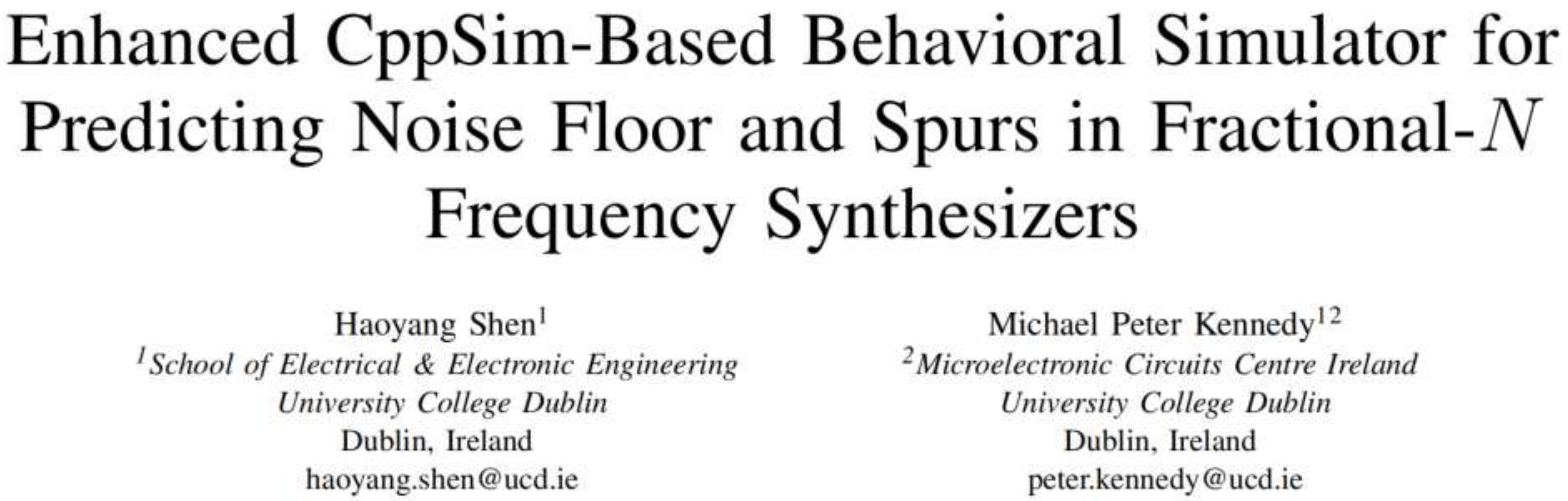
Tutorial on CppSim-based Fractional-N CP-PLL Design

Dr. Haoyang Shen

Circuits and Systems Group

11/02/2026

Fractional-N CP-PLL Model



Enhanced CppSim-Based Behavioral Simulator for Predicting Noise Floor and Spurs in Fractional-N Frequency Synthesizers

Haoyang Shen[1]
[1] *School of Electrical & Electronic Engineering*
University College Dublin
Dublin, Ireland
haoyang.shen@ucd.ie

Michael Peter Kennedy[1 2]
[2] *Microelectronic Circuits Centre Ireland*
University College Dublin
Dublin, Ireland
peter.kennedy@ucd.ie

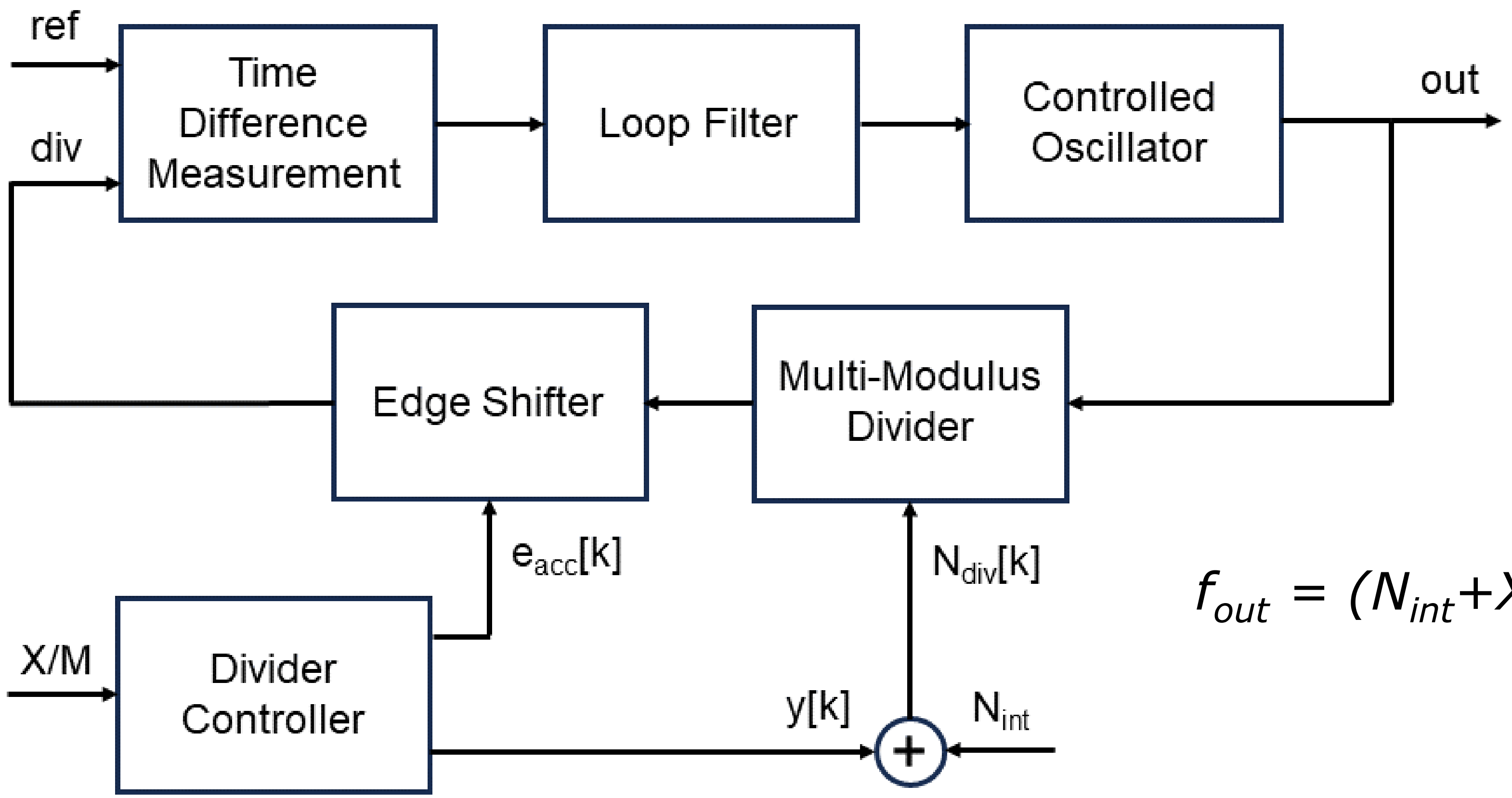
H. Shen and M. P. Kennedy, "Enhanced CppSim-Based Behavioral Simulator for Predicting Noise Floor and Spurs in Fractional-N Frequency Synthesizers," 2025 IEEE International Symposium on Circuits and Systems (ISCAS), London, United Kingdom, 2025, pp. 1-5,
doi: 10.1109/ISCAS56072.2025.11043584.

Outlines

- ☐ Fractional-N Frequency Synthesizer Model
- ☐ CppSim SUE 2 Introduction
- ☐ Extracting Simulink S-function Model
- ☐ MATLAB Code

Fractional-N Frequency Synthesizer Model

Fractional-*N* Frequency Synthesizer



$$f_{out} = (N_{int}+X/M)f_{ref}$$

Fractional-*N* facilitates a high reference frequency and small frequency steps

Taighde Éireann Research Ireland

Key Variables

Nominal divide ratio:

$N_{nom} = (N_{int}+X/M)$

Instantaneous divide ratio:

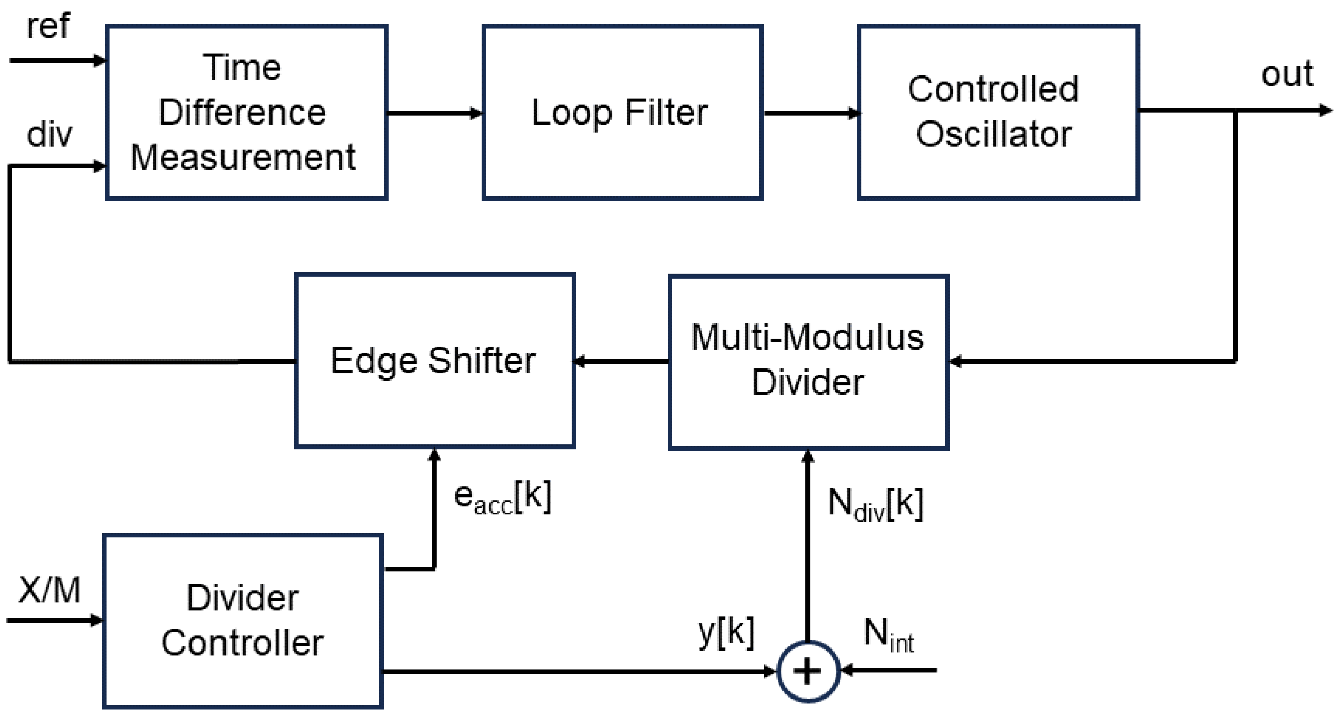
$N_{div}[k] = (N_{int}+y[k])$

Divider quantization error:

$e[k] = N_{div}[k] - N_{nom}$
$= y[k] - X/M$

Accumulated quantization error (proportional to phase/time difference):

$e_{acc}[k] = \Sigma_{i=0}\, e[i]$

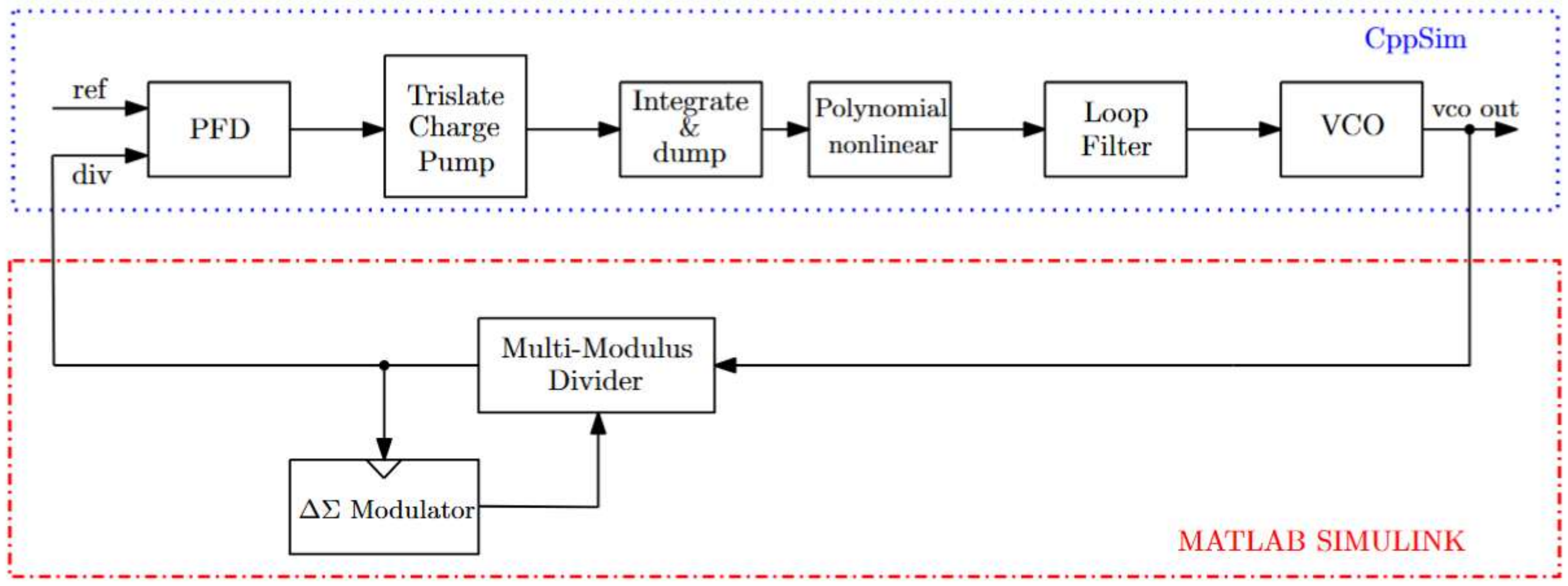


$f_{out} = (N_{int}+X/M)f_{ref}$

Taighde Éireann Research Ireland

CP-PLL Model

- CppSim
- Simulink



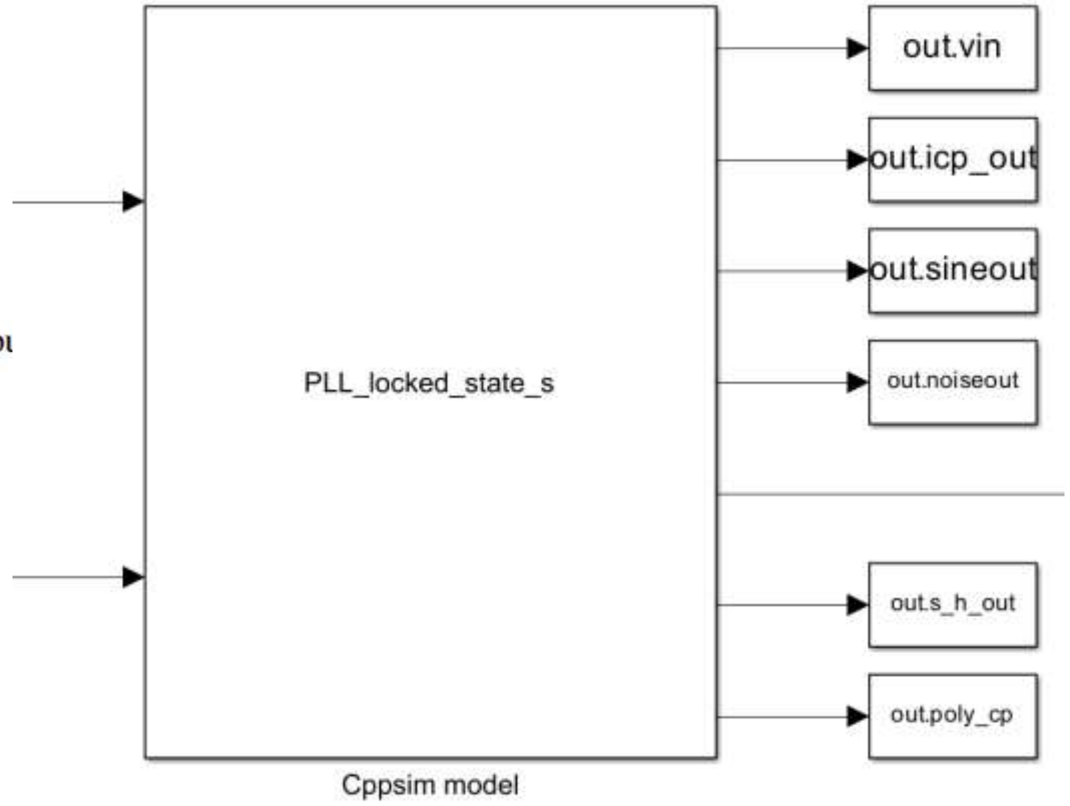
S-function from CppSim

The input ports:

1) charge pump offset
2) integer divide value

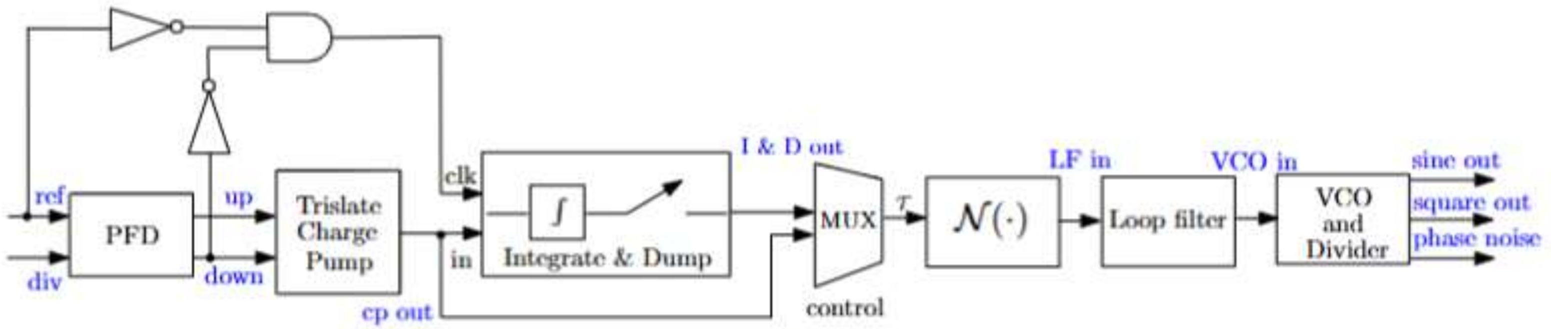
The output ports:

1) VCO input
2) charge pump output
3) VCO and divider output (sine signal)
4) PLL phase noise output
5) VCO and divider output (square signal, the *div* signal as the PFD inpu
6) Integrate & dump output
7) loop filter input

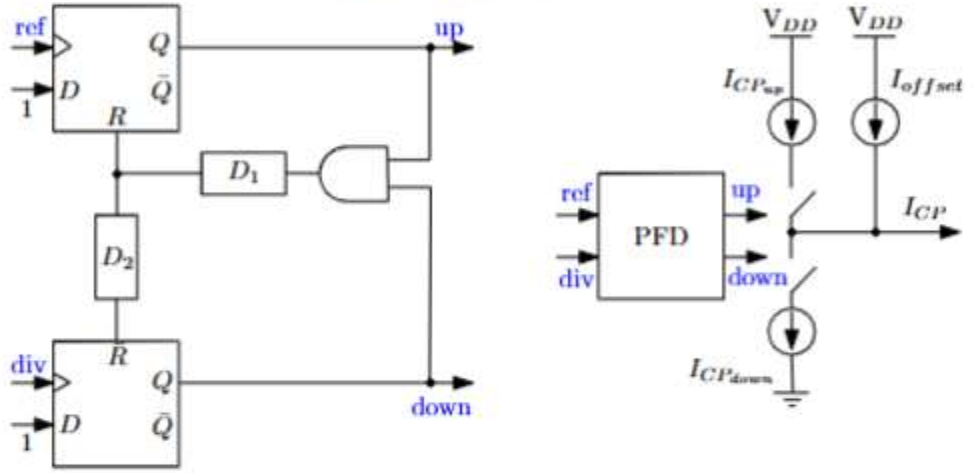


Taighde Éireann
Research Ireland

Model details



(a) CppSim circuits

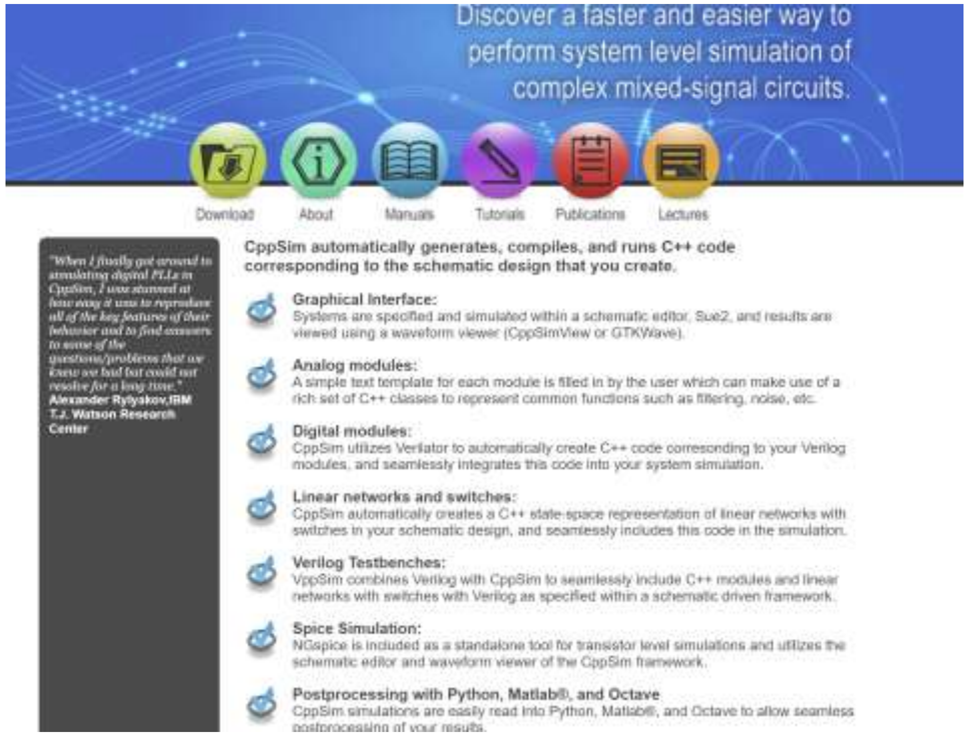


(b) The details of PFD and CP

CppSim SUE 2

CppSim SUE2

- Graphic interface for construction of PLL using elements from built-in libraries
- CppSim tutorial is available: https://www.cppsim.com/manuals.html


Discover a faster and easier way to perform system level simulation of complex mixed-signal circuits.
Download
About
Manuals
Tutorials
Publications
Lectures

"When I finally got around to simulating digital PLLs in CppSim, I was stunned at how easy it was to reproduce all of the key features of their behavior and to find answers to some of the questions/problems that we knew we had but could not resolve for a long time."
Alexander Rylyakov, IBM T.J. Watson Research Center

CppSim automatically generates, compiles, and runs C++ code corresponding to the schematic design that you create.

Graphical Interface:
Systems are specified and simulated within a schematic editor, Sue2, and results are viewed using a waveform viewer (CppSimView or GTKWave).

Analog modules:
A simple text template for each module is filled in by the user which can make use of a rich set of C++ classes to represent common functions such as filtering, noise, etc.

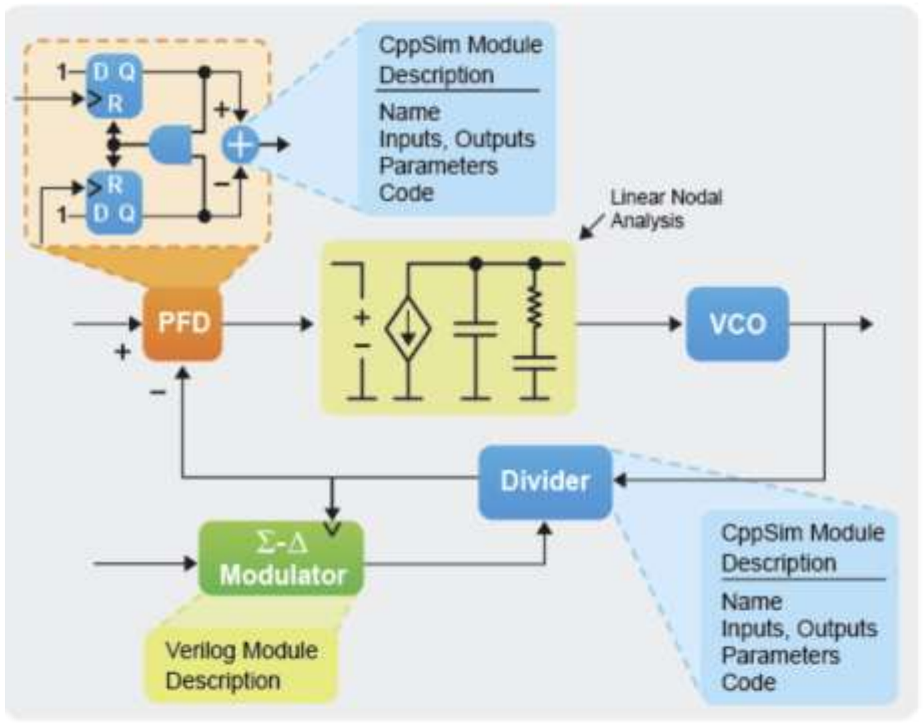
Digital modules:
CppSim utilizes Verilator to automatically create C++ code corresonding to your Verilog modules, and seamlessly integrates this code into your system simulation.

Linear networks and switches:
CppSim automatically creates a C++ state-space representation of linear networks with switches in your schematic design, and seamlessly includes this code in the simulation.

Verilog Testbenches:
VppSim combines Verilog with CppSim to seamlessly include C++ modules and linear networks with switches with Verilog as specified within a schematic driven framework.

Spice Simulation:
NGspice is included as a standalone tool for transistor level simulations and utilizes the schematic editor and waveform viewer of the CppSim framework.

Postprocessing with Python, Matlab®, and Octave
CppSim simulations are easily read into Python, Matlab®, and Octave to allow seamless postprocessing of your results.

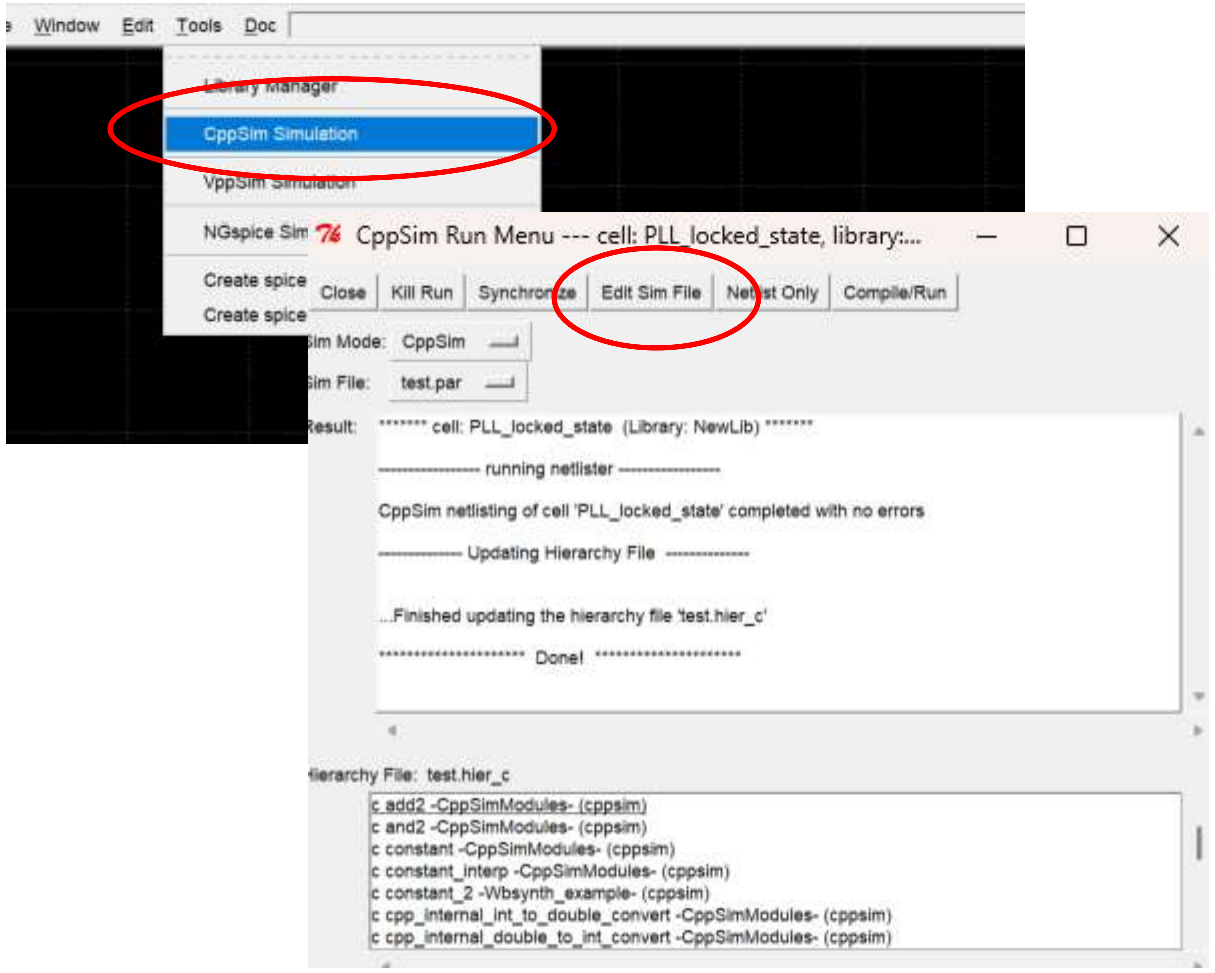


Extracting Simulink S-function Model

Taighde Éireann Research Ireland

Extracting Simulink S-function

- ‘CppSim Simulation’ under ‘Tool’ opens an interface
- Select ‘Edit Sim File’ to open the simulation file for the model to be exported



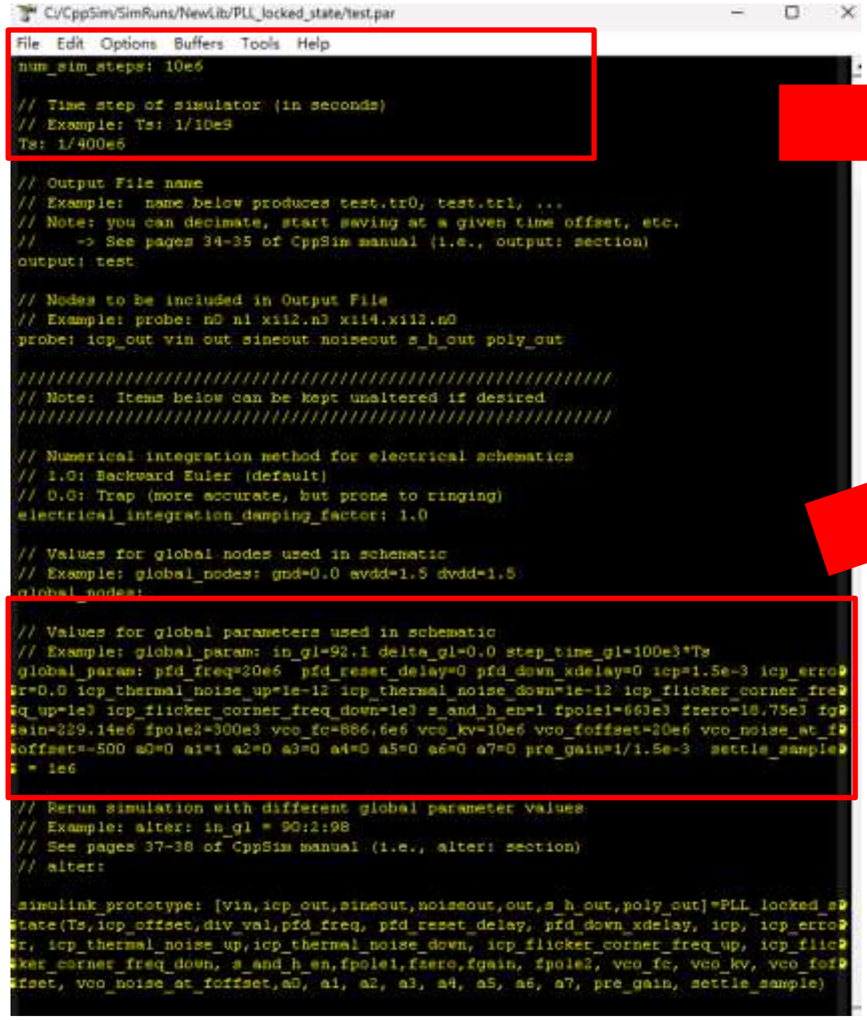
Sim Mode: CppSim
Sim File: test.par
Result:

```
******* cell: PLL_locked_state  (Library: NewLib) *******

--------------- running netlister ---------------

CppSim netlisting of cell 'PLL_locked_state' completed with no errors

------------ Updating Hierarchy File ------------


...Finished updating the hierarchy file 'test.hier_c'

******************** Done! ********************
```

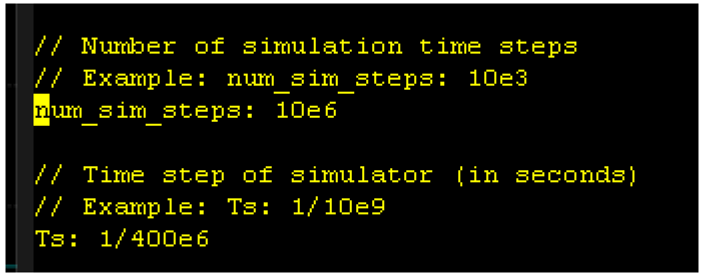
Hierarchy File: test.hier_c

```
c add2 -CppSimModules- (cppsim)
c and2 -CppSimModules- (cppsim)
c constant -CppSimModules- (cppsim)
c constant_interp -CppSimModules- (cppsim)
c constant_2 -Wbsynth_example- (cppsim)
c cpp_internal_int_to_double_convert -CppSimModules- (cppsim)
c cpp_internal_double_to_int_convert -CppSimModules- (cppsim)
```

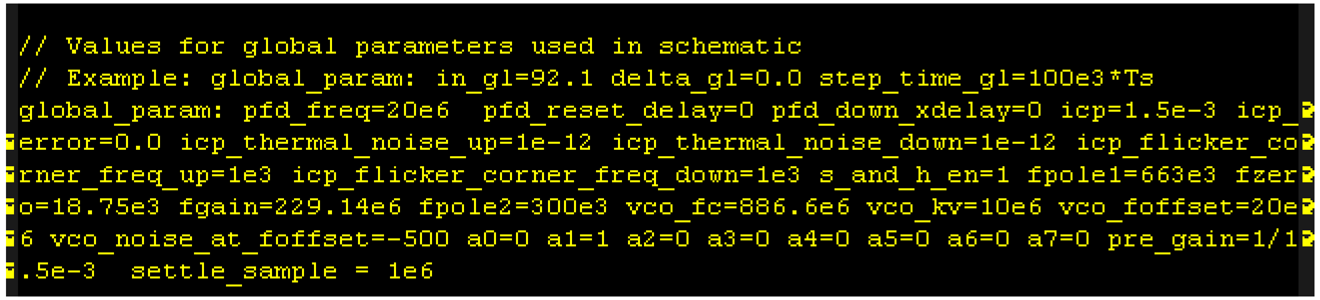
Extracting Simulink S-function


C:/CppSim/SimRuns/NewLib/PLL_locked_state/test.par
File Edit Options Buffers Tools Help

```
num_sim_steps: 10e6

// Time step of simulator (in seconds)
// Example: Ts: 1/10e9
Ts: 1/400e6

// Output File name
// Example:  name below produces test.tr0, test.tr1, ...
// Note: you can decimate, start saving at a given time offset, etc.
//     -> See pages 34-35 of CppSim manual (i.e., output: section)
output: test

// Nodes to be included in Output File
// Example: probe: n0 n1 xi12.n3 xi14.xi12.n0
probe: icp_out vin out sineout noiseout s_h_out poly_out

/////////////////////////////////////////////////////////////////
// Note:  Items below can be kept unaltered if desired
/////////////////////////////////////////////////////////////////

// Numerical integration method for electrical schematics
// 1.0: Backward Euler (default)
// 0.0: Trap (more accurate, but prone to ringing)
electrical_integration_damping_factor: 1.0

// Values for global nodes used in schematic
// Example: global_nodes: gnd=0.0 avdd=1.5 dvdd=1.5
global_nodes:

// Values for global parameters used in schematic
// Example: global_param: in_gl=92.1 delta_gl=0.0 step_time_gl=100e3*Ts
global_param: pfd_freq=20e6  pfd_reset_delay=0 pfd_down_xdelay=0 icp=1.5e-3 icp_erro
r=0.0 icp_thermal_noise_up=1e-12 icp_thermal_noise_down=1e-12 icp_flicker_corner_fre
q_up=1e3 icp_flicker_corner_freq_down=1e3 s_and_h_en=1 fpole1=663e3 fzero=18.75e3 fg
ain=229.14e6 fpole2=300e3 vco_fc=886.6e6 vco_kv=10e6 vco_foffset=20e6 vco_noise_at_f
offset=-500 a0=0 a1=1 a2=0 a3=0 a4=0 a5=0 a6=0 a7=0 pre_gain=1/1.5e-3  settle_sample
 = 1e6

// Rerun simulation with different global parameter values
// Example: alter: in_gl = 90:2:98
// See pages 37-38 of CppSim manual (i.e., alter: section)
// alter:

simulink_prototype: [vin,icp_out,sineout,noiseout,out,s_h_out,poly_out]=PLL_locked_s
tate(Ts,icp_offset,div_val,pfd_freq, pfd_reset_delay, pfd_down_xdelay, icp, icp_erro
r, icp_thermal_noise_up,icp_thermal_noise_down, icp_flicker_corner_freq_up, icp_flic
ker_corner_freq_down, s_and_h_en,fpole1,fzero,fgain, fpole2, vco_fc, vco_kv, vco_fof
fset, vco_noise_at_foffset,a0, a1, a2, a3, a4, a5, a6, a7, pre_gain, settle_sample)
```



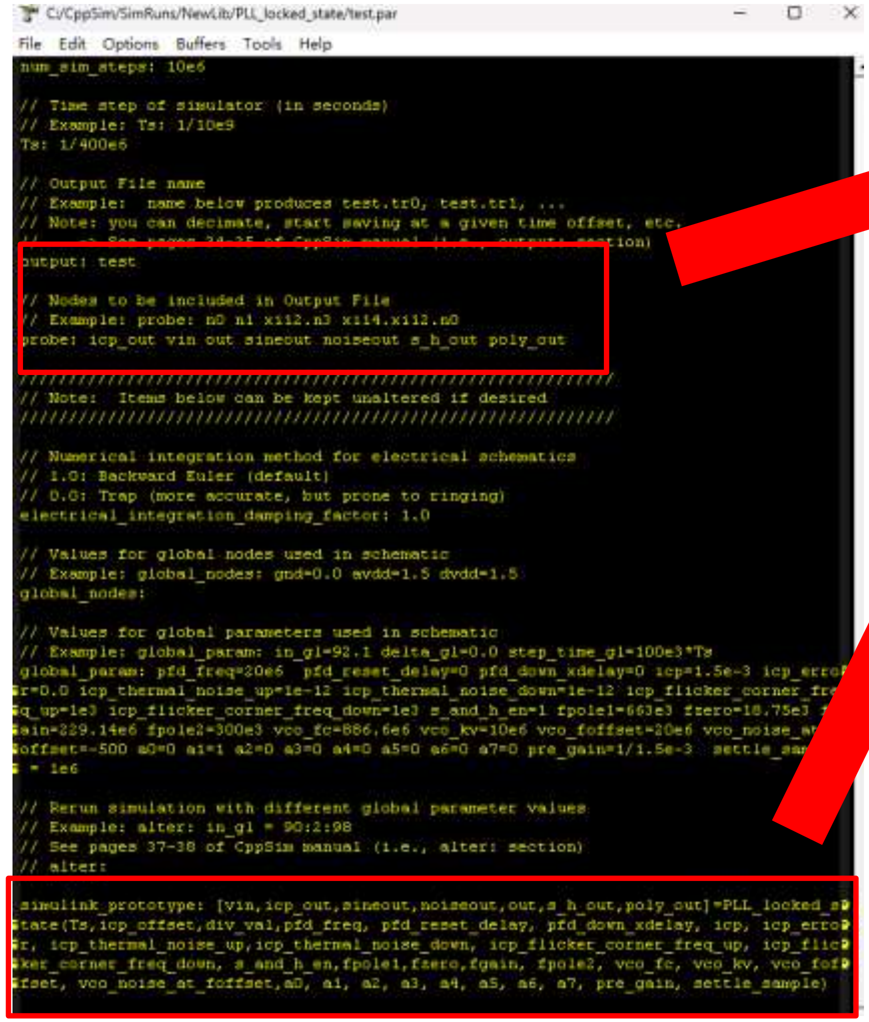
```
// Number of simulation time steps
// Example: num_sim_steps: 10e3
num_sim_steps: 10e6

// Time step of simulator (in seconds)
// Example: Ts: 1/10e9
Ts: 1/400e6
```



```
// Values for global parameters used in schematic
// Example: global_param: in_gl=92.1 delta_gl=0.0 step_time_gl=100e3*Ts
global_param: pfd_freq=20e6  pfd_reset_delay=0 pfd_down_xdelay=0 icp=1.5e-3 icp_
error=0.0 icp_thermal_noise_up=1e-12 icp_thermal_noise_down=1e-12 icp_flicker_co
rner_freq_up=1e3 icp_flicker_corner_freq_down=1e3 s_and_h_en=1 fpole1=663e3 fzer
o=18.75e3 fgain=229.14e6 fpole2=300e3 vco_fc=886.6e6 vco_kv=10e6 vco_foffset=20e
6 vco_noise_at_foffset=-500 a0=0 a1=1 a2=0 a3=0 a4=0 a5=0 a6=0 a7=0 pre_gain=1/1
.5e-3  settle_sample = 1e6
```

- Given default Ts and sampling numbers; actual values are given in MATLAB Simulink
- Parameters are given specific values in the sim file: values are dummy; actual values are given in MATLAB Simulink simulation

Extracting Simulink S-function


C:/CppSim/SimRuns/NewLib/PLL_locked_state/test.par
File Edit Options Buffers Tools Help

```
num_sim_steps: 10e6

// Time step of simulator (in seconds)
// Example: Ts: 1/10e9
Ts: 1/400e6

// Output File name
// Example:  name below produces test.tr0, test.tr1, ...
// Note: you can decimate, start saving at a given time offset, etc.
//     -> See pages 34-35 of CppSim manual (i.e., output: section)
output: test

// Nodes to be included in Output File
// Example: probe: n0 n1 xi12.n3 xi14.xi12.n0
probe: icp_out vin out sineout noiseout s_h_out poly_out

//////////////////////////////////////////////////////////////
// Note:  Items below can be kept unaltered if desired
//////////////////////////////////////////////////////////////

// Numerical integration method for electrical schematics
// 1.0: Backward Euler (default)
// 0.0: Trap (more accurate, but prone to ringing)
electrical_integration_damping_factor: 1.0

// Values for global nodes used in schematic
// Example: global_nodes: gnd=0.0 avdd=1.5 dvdd=1.5
global_nodes:

// Values for global parameters used in schematic
// Example: global_param: in_gl=92.1 delta_gl=0.0 step_time_gl=100e3*Ts
global_param: pfd_freq=20e6  pfd_reset_delay=0 pfd_down_xdelay=0 icp=1.5e-3 icp_erro
r=0.0 icp_thermal_noise_up=1e-12 icp_thermal_noise_down=1e-12 icp_flicker_corner_fre
q_up=1e3 icp_flicker_corner_freq_down=1e3 s_and_h_en=1 fpole1=663e3 fzero=18.75e3 f
ain=229.14e6 fpole2=300e3 vco_fc=886.6e6 vco_kv=10e6 vco_foffset=20e6 vco_noise_at
offset=-500 a0=0 a1=1 a2=0 a3=0 a4=0 a5=0 a6=0 a7=0 pre_gain=1/1.5e-3  settle_sam
 = 1e6

// Rerun simulation with different global parameter values
// Example: alter: in_gl = 90:2:98
// See pages 37-38 of CppSim manual (i.e., alter: section)
// alter:

simulink_prototype: [vin,icp_out,sineout,noiseout,out,s_h_out,poly_out]=PLL_locked_s
tate(Ts,icp_offset,div_val,pfd_freq, pfd_reset_delay, pfd_down_xdelay, icp, icp_erro
r, icp_thermal_noise_up,icp_thermal_noise_down, icp_flicker_corner_freq_up, icp_flic
ker_corner_freq_down, s_and_h_en,fpole1,fzero,fgain, fpole2, vco_fc, vco_kv, vco_fof
fset, vco_noise_at_foffset,a0, a1, a2, a3, a4, a5, a6, a7, pre_gain, settle_sample)
```

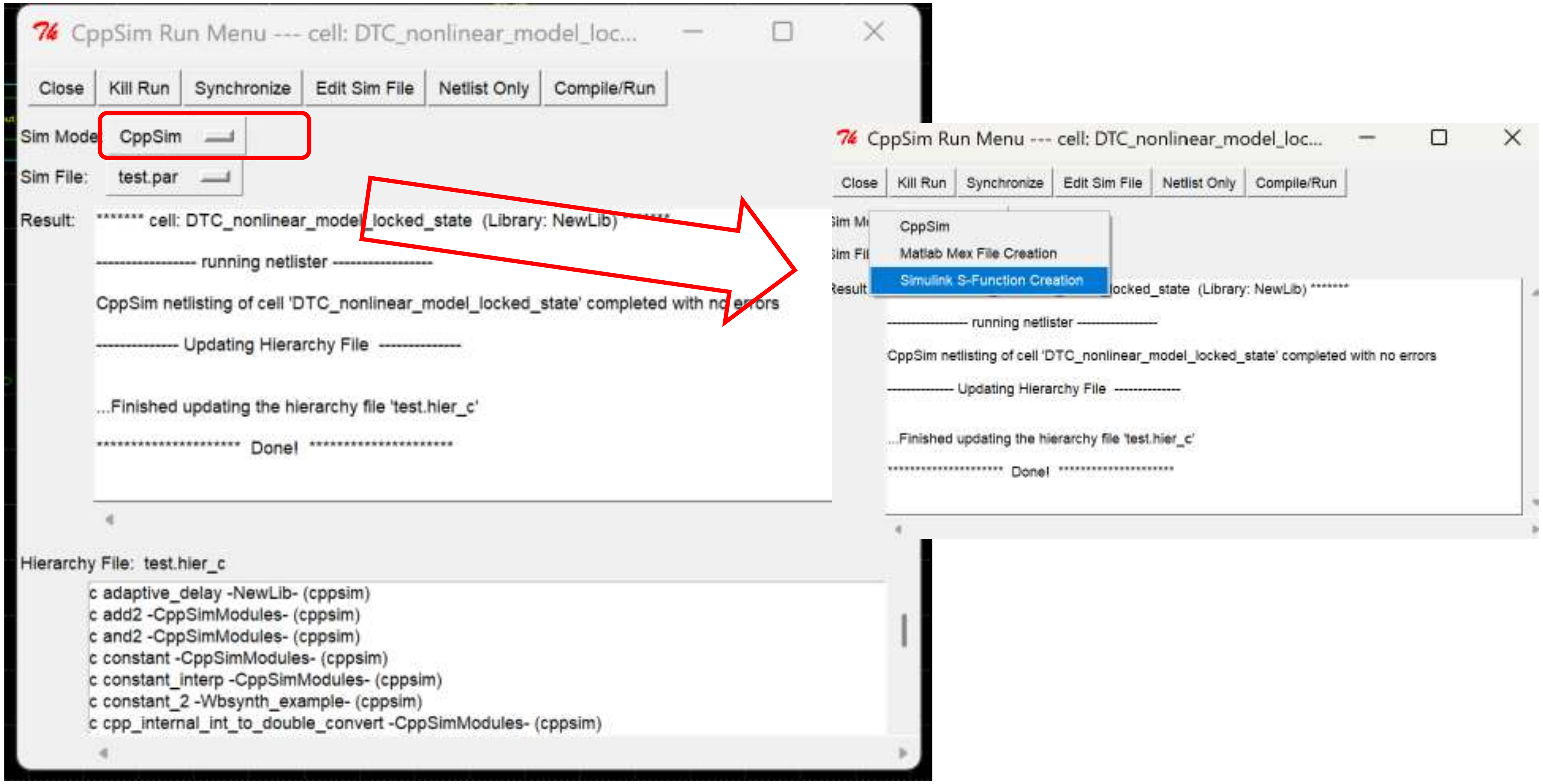
```
// Nodes to be included in Output File
// Example: probe: n0 n1 xi12.n3 xi14.xi12.n0
probe: icp_out vin out sineout noiseout s_h_out poly_out
```

- Define the probe for the output in MATLAB Simulink

```
 simulink_prototype: [vin,icp_out,sineout,noiseout,out,s_h_out,poly_out]=PLL_lock
ed_state(Ts,icp_offset,div_val,pfd_freq, pfd_reset_delay, pfd_down_xdelay, icp, 
icp_error, icp_thermal_noise_up,icp_thermal_noise_down, icp_flicker_corner_freq_
up, icp_flicker_corner_freq_down, s_and_h_en,fpole1,fzero,fgain, fpole2, vco_fc,
 vco_kv, vco_foffset, vco_noise_at_foffset,a0, a1, a2, a3, a4, a5, a6, a7, pre_g
ain, settle_sample)
```

- Syntax'simulink_prototype:[out1,out2,…]=func_name(in1,in2,…,param1,param2,…)'
- Notice that sampling period 'Ts' has to be included in parameters, just make it the first
- List as shown in the screenshot; However, number of simulation steps is not required here
- All parameters listed should be included
- Input from Simulink, e.g., cp_offset, divider controller output, should be listed (icp_offset and div_val)

Extracting Simulink S-function



Sim Mode: CppSim

Sim File: test.par

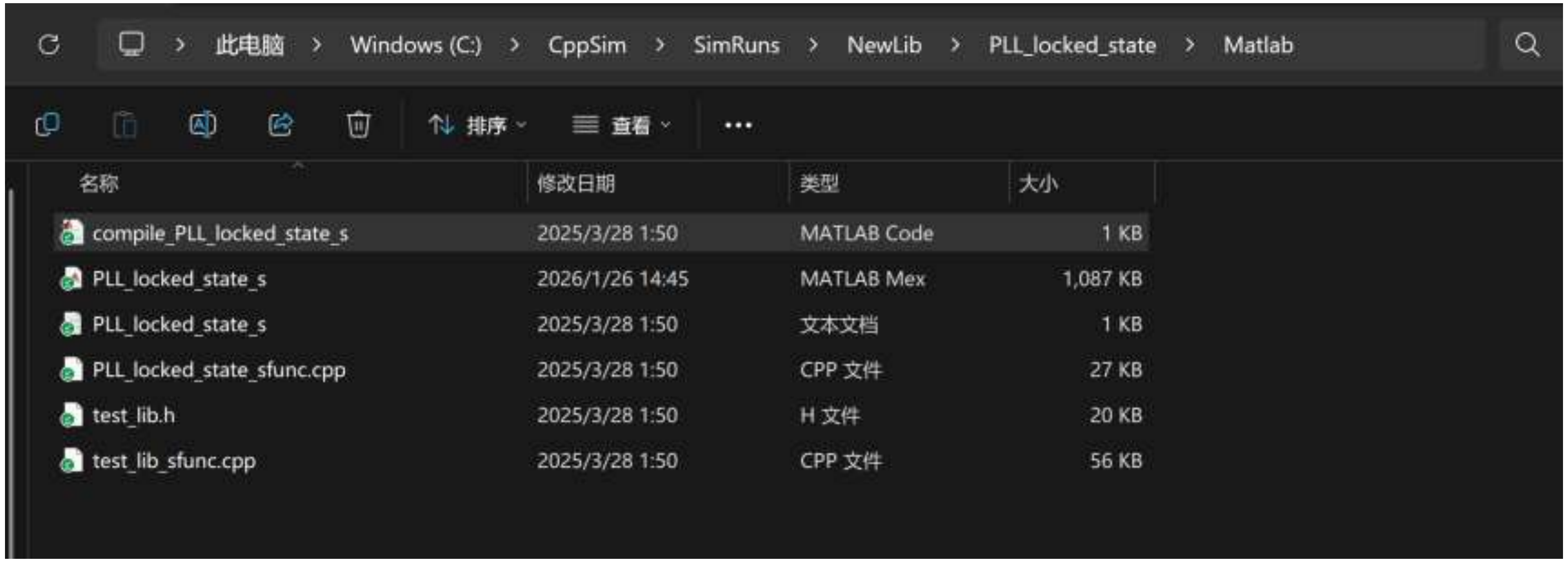
Result:

```
******* cell: DTC_nonlinear_model_locked_state  (Library: NewLib) *******

----------------- running netlister -----------------

CppSim netlisting of cell 'DTC_nonlinear_model_locked_state' completed with no errors

-------------- Updating Hierarchy File --------------


...Finished updating the hierarchy file 'test.hier_c'

********************** Done! **********************
```

Hierarchy File: test.hier_c

```
c adaptive_delay -NewLib- (cppsim)
c add2 -CppSimModules- (cppsim)
c and2 -CppSimModules- (cppsim)
c constant -CppSimModules- (cppsim)
c constant_interp -CppSimModules- (cppsim)
c constant_2 -Wbsynth_example- (cppsim)
c cpp_internal_int_to_double_convert -CppSimModules- (cppsim)
```

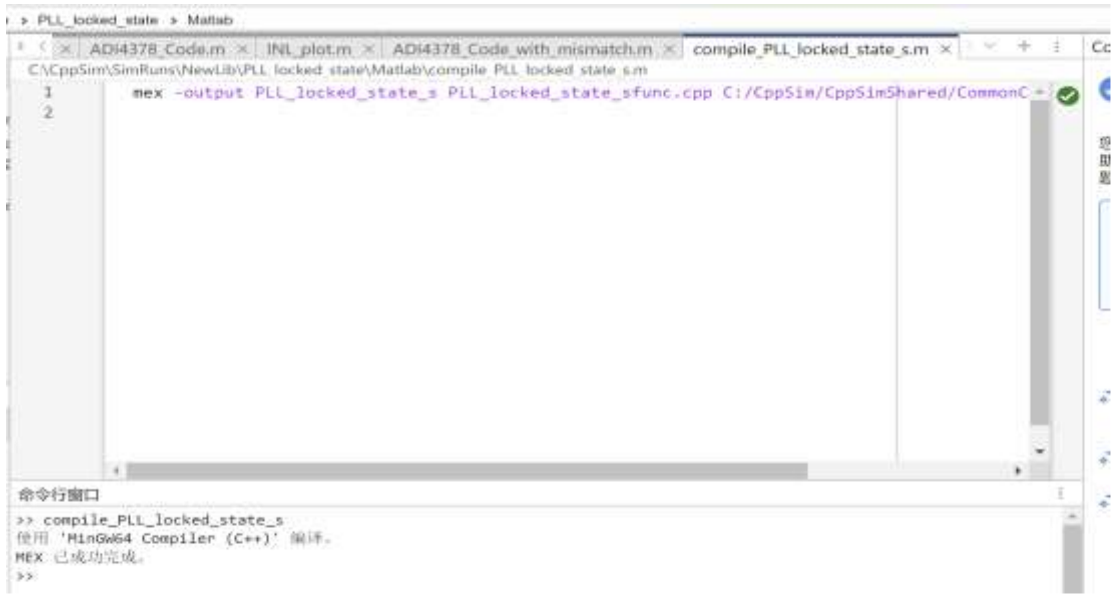
Extracting Simulink S-function


此电脑 > Windows (C:) > CppSim > SimRuns > NewLib > PLL_locked_state > Matlab
排序
查看

名称	修改日期	类型	大小
compile_PLL_locked_state_s	2025/3/28 1:50	MATLAB Code	1 KB
PLL_locked_state_s	2026/1/26 14:45	MATLAB Mex	1,087 KB
PLL_locked_state_s	2025/3/28 1:50	文本文档	1 KB
PLL_locked_state_sfunc.cpp	2025/3/28 1:50	CPP 文件	27 KB
test_lib.h	2025/3/28 1:50	H 文件	20 KB
test_lib_sfunc.cpp	2025/3/28 1:50	CPP 文件	56 KB

- If extraction is successful, a series of files will be generated in 'C:\CppSim\SimRuns\*Libraryname**Modelname*\Matlab'
- 'compile_*Modelname*_s.m' is the MATLAB script for compilation of the model from C++
- '*Modelname*_s.txt' contains parameters, input and output information

Extracting Simulink S-function

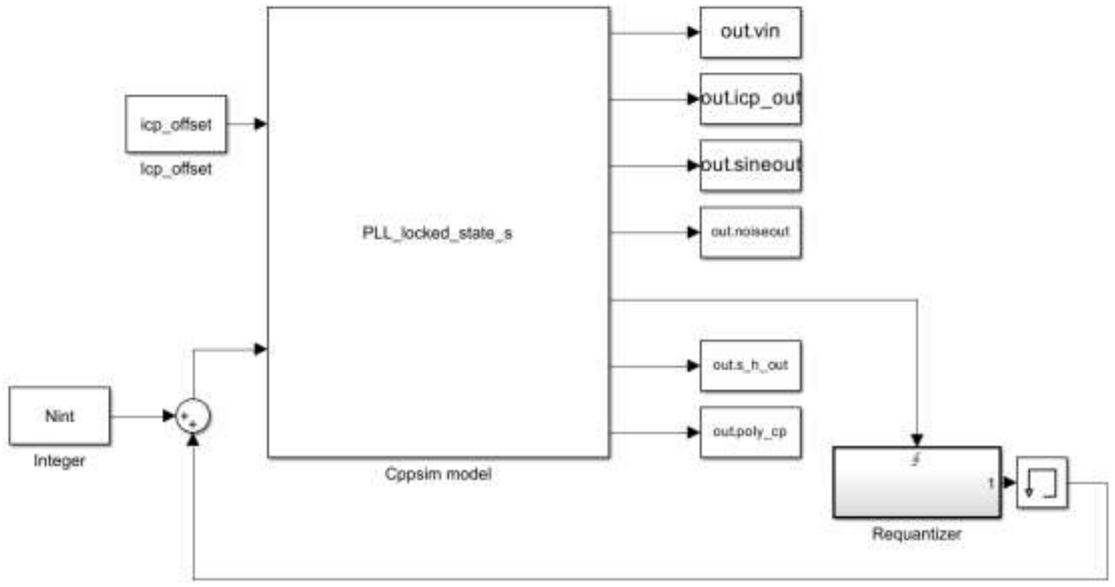

PLL_locked_state > Matlab
ADI4378_Code.m
INL_plot.m
ADI4378_Code_with_mismatch.m
compile_PLL_locked_state_s.m
C:\CppSim\SimRuns\NewLib\PLL_locked_state\Matlab\compile_PLL_locked_state_s.m
命令行窗口

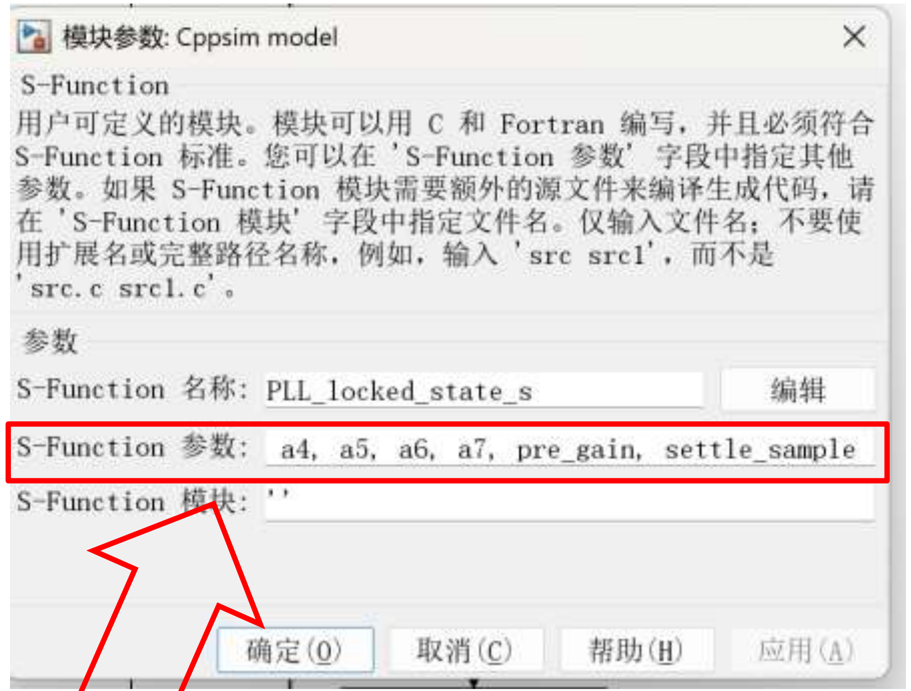
```
mex -output PLL_locked_state_s PLL_locked_state_sfunc.cpp C:/CppSim/CppSimShared/CommonC
```

```
>> compile_PLL_locked_state_s
使用 'MinGW64 Compiler (C++)' 编译。
MEX 已成功完成。
>>
```

- To run the MATLAB script, C++ compiler for MATLAB should be installed, e.g., mingw64
- Successful compilation gives a '.mexw64' file

Extracting Simulink S-function





模块参数: Cppsim model

S-Function

用户可定义的模块。模块可以用 C 和 Fortran 编写，并且必须符合 S-Function 标准。您可以在 'S-Function 参数' 字段中指定其他参数。如果 S-Function 模块需要额外的源文件来编译生成代码，请在 'S-Function 模块' 字段中指定文件名。仅输入文件名；不要使用扩展名或完整路径名称，例如，输入 'src src1'，而不是 'src.c src1.c'。

参数

S-Function 名称: PLL_locked_state_s 编辑

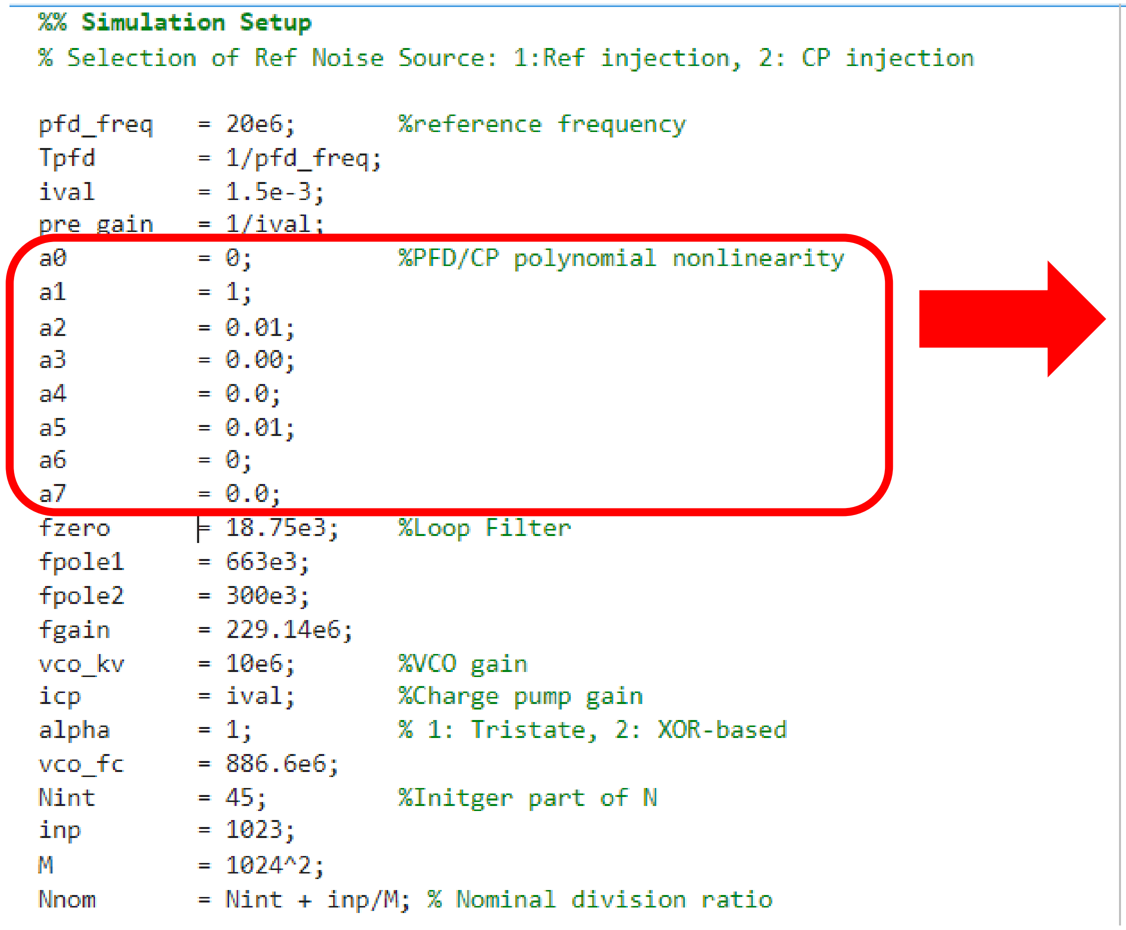
S-Function 参数: a4, a5, a6, a7, pre_gain, settle_sample

S-Function 模块: ''

```
--- Simulink Block 'PLL_locked_state_s' ---
Parameters:  Ts, pfd_freq, pfd_reset_delay, pfd_down_xdelay, icp, icp_error, icp_thermal_noise_up, icp_thermal_noise_down, icp_flicker_corner_freq_up,
icp_flicker_corner_freq_down, s_and_h_en, fpole1, fzero, fgain, fpole2, vco_fc, vco_kv, vco_foffset, vco_noise_at_foffset, a0, a1, a2, a3, a4, a5, a6, a7,
pre_gain, settle_sample
Inputs:  icp_offset, div_val
Outputs:  vin, icp_out, sineout, noiseout, out, s_h_out, poly_out
```

MATLAB Code

PLL basic setting

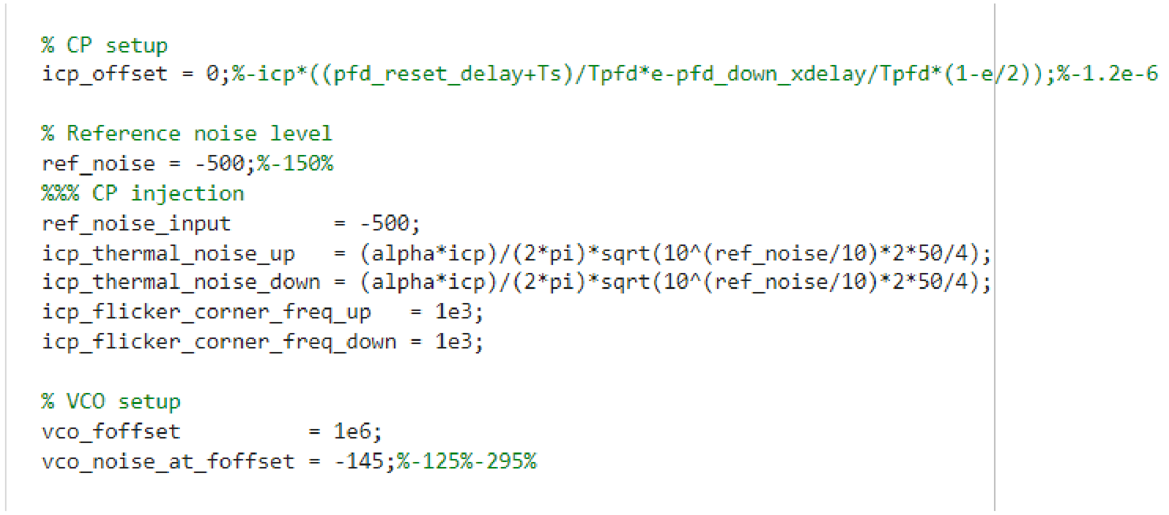


```
%% Simulation Setup
% Selection of Ref Noise Source: 1:Ref injection, 2: CP injection

pfd_freq     = 20e6;       %reference frequency
Tpfd         = 1/pfd_freq;
ival         = 1.5e-3;
pre_gain     = 1/ival;
a0           = 0;          %PFD/CP polynomial nonlinearity
a1           = 1;
a2           = 0.01;
a3           = 0.00;
a4           = 0.0;
a5           = 0.01;
a6           = 0;
a7           = 0.0;
fzero        = 18.75e3;    %Loop Filter
fpole1       = 663e3;
fpole2       = 300e3;
fgain        = 229.14e6;
vco_kv       = 10e6;       %VCO gain
icp          = ival;       %Charge pump gain
alpha        = 1;          % 1: Tristate, 2: XOR-based
vco_fc       = 886.6e6;
Nint         = 45;         %Initger part of N
inp          = 1023;
M            = 1024^2;
Nnom         = Nint + inp/M; % Nominal division ratio
```

PFD/CP polynomial nonlinearity

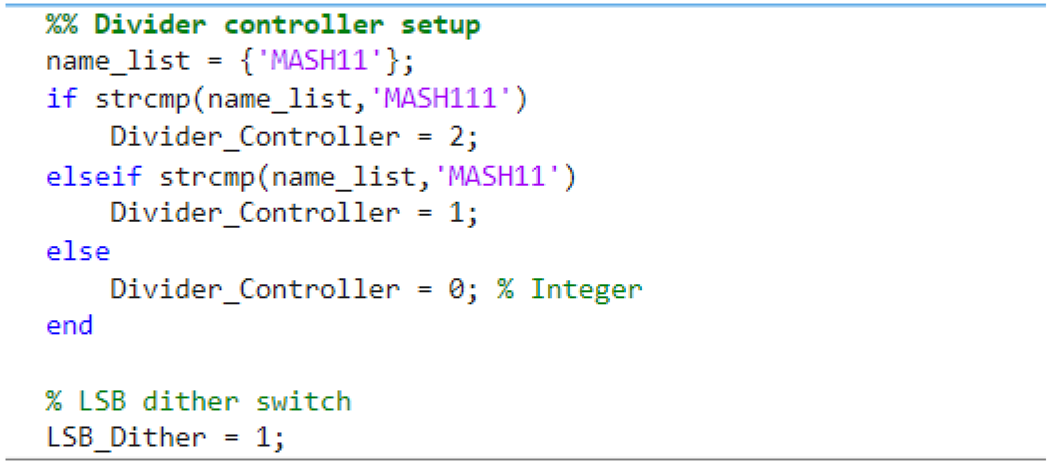
$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \dots$$

Noise and Divider controller



```
% CP setup
icp_offset = 0;%-icp*((pfd_reset_delay+Ts)/Tpfd*e-pfd_down_xdelay/Tpfd*(1-e/2));%-1.2e-6

% Reference noise level
ref_noise = -500;%-150%
%%% CP injection
ref_noise_input        = -500;
icp_thermal_noise_up   = (alpha*icp)/(2*pi)*sqrt(10^(ref_noise/10)*2*50/4);
icp_thermal_noise_down = (alpha*icp)/(2*pi)*sqrt(10^(ref_noise/10)*2*50/4);
icp_flicker_corner_freq_up   = 1e3;
icp_flicker_corner_freq_down = 1e3;

% VCO setup
vco_foffset          = 1e6;
vco_noise_at_foffset = -145;%-125%-295%
```

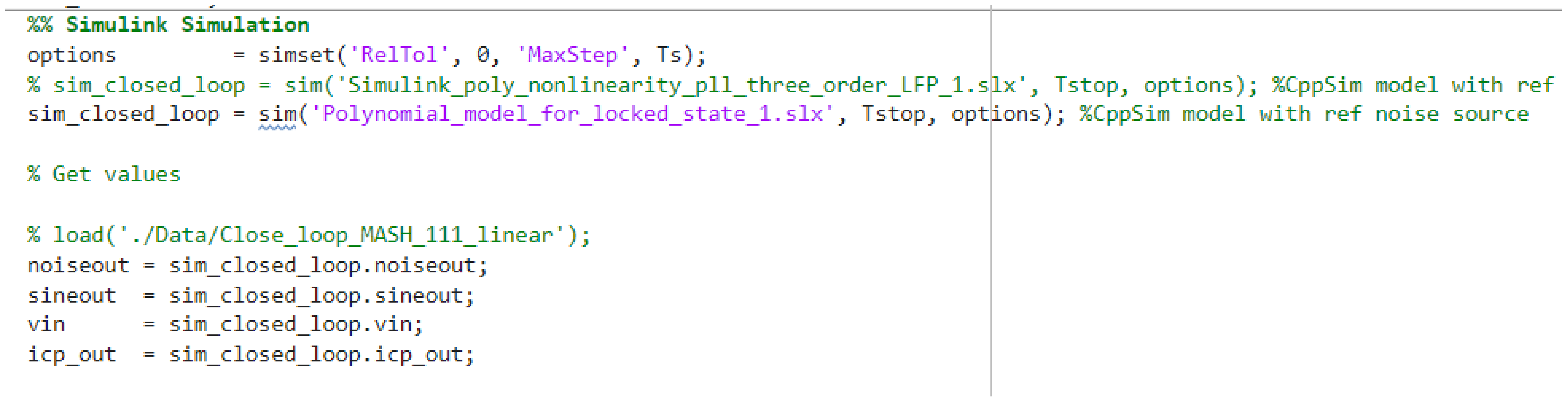
Noise setting including reference noise, CP noise, and VCO noise



```
%% Divider controller setup
name_list = {'MASH11'};
if strcmp(name_list,'MASH111')
    Divider_Controller = 2;
elseif strcmp(name_list,'MASH11')
    Divider_Controller = 1;
else
    Divider_Controller = 0; % Integer
end

% LSB dither switch
LSB_Dither = 1;
```

MASH 1-1 and MASH 1-1-1 can be used

Model Running



```
%% Simulink Simulation
options          = simset('RelTol', 0, 'MaxStep', Ts);
% sim_closed_loop = sim('Simulink_poly_nonlinearity_pll_three_order_LFP_1.slx', Tstop, options); %CppSim model with ref
sim_closed_loop = sim('Polynomial_model_for_locked_state_1.slx', Tstop, options); %CppSim model with ref noise source

% Get values

% load('./Data/Close_loop_MASH_111_linear');
noiseout = sim_closed_loop.noiseout;
sineout  = sim_closed_loop.sineout;
vin      = sim_closed_loop.vin;
icp_out  = sim_closed_loop.icp_out;
```

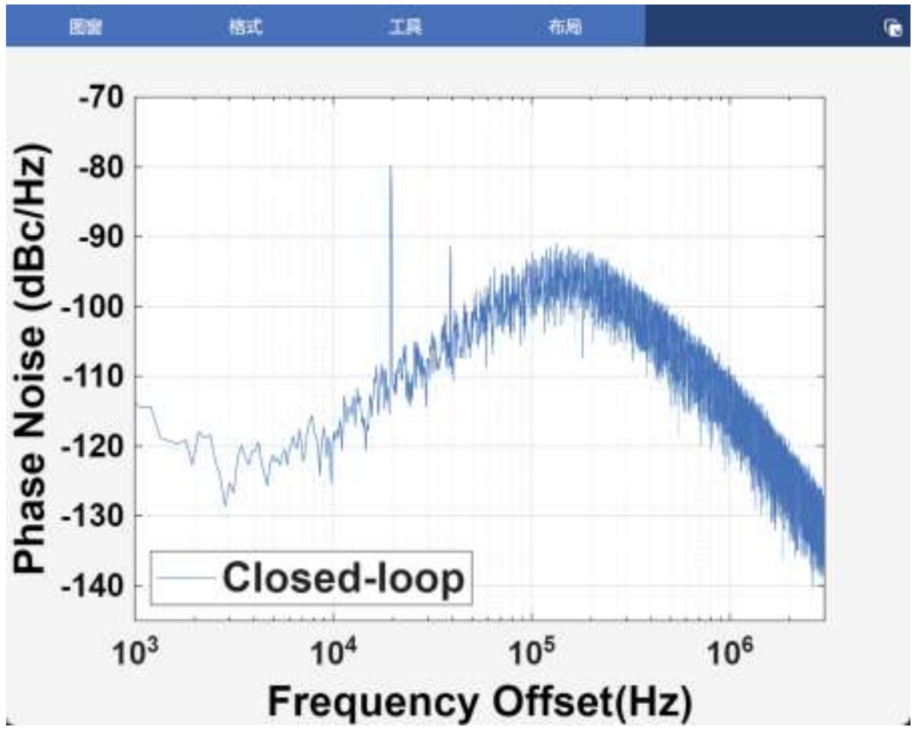
Taighde Éireann Research Ireland

Output Phase Noise

- ☐ The noise output of 'vco_and_divider_with_noise' block is frequency error of the output signal. Note that this is not phase noise
- ☐ In order to find the phase noise, the output noise should be accumulated:

```
% Phase noise calculation
Kv = 1;    % VCO gain (Hz/V) - this is one for noiseout from CppSim
phase_noise = filter(2*pi*Kv/Fs,[1 -1],noise); %running sum of noise
```

- ☐ The original MATLAB script can be found in 'C:\CppSim\CppSimShared\MatlabCode\plot_pll_phasenoise'
- ☐ Used to generate power spectral density of the phase noise



Open Access

The code and model is open access on Github

https://github.com/NKdeNY1996/Fractional_N_PLL_CppSim

Please ensure that our work is properly cited and used strictly for research and verification purposes.

Don’t forget give us a star on GitHub if you like our work.

Cite This: H. Shen and M. P. Kennedy, "Enhanced CppSim-Based Behavioral Simulator for Predicting Noise Floor and Spurs in Fractional-N Frequency Synthesizers," 2025 IEEE International Symposium on Circuits and Systems (ISCAS), London, United Kingdom, 2025, pp. 1-5, doi: 10.1109/ISCAS56072.2025.11043584