
Table of Contents

.....	1
Initialization	1
Read from stimulus file	1
Perform inverse sqrt on STIM	2
Read from ModelSIM output file	3
Verification	4

`% Test and verification script`

Initialization

```
% Generics
w_bits = 32;
f_bits = 16;
diff = w_bits - f_bits;
N_iterations = 3;

% Fimath properties
Fm = fimath('RoundingMethod','Nearest',...
'OverflowAction','Wrap',...
'ProductWordLength',w_bits,...
'ProductFractionLength',f_bits,...
'SumMode','SpecifyPrecision',...
'SumWordLength',w_bits,...
'SumFractionLength',f_bits);
```

Read from stimulus file

```
fileID = fopen('stim.txt','r');

a = [];

% Convert fixed point bit-string to decimal value
for i = 1:100

    float = 0;
    int = 0;
    frac = 0;

    line_a = fgetl(fileID);

    % Calculate integer component
    for i = 1:diff
        int = int + str2num(line_a(i))*2^(diff-i);
    end

    % Calculate fractional component
```

```

    for i = diff+1:w_bits
        frac = frac + str2num(line_a(i))/(2^(i-diff));
    end

    float = int + frac;

    % Convert to fixed-point datatype
    fixed = fi(float,0,w_bits,f_bits);

    a = [a fixed];

end
fclose(fileID);

```

Perform inverse sqrt on STIM

```

testbench = [];

y0 = 1;

% Emulate rsqrt calculation made in ModelSim
for i = 1:100

    vector = bin(a(i));

    % Count leading zeros

    check = 0;
    lzc = 0;

    for j = 1:diff
        if(vector(j) == '1')
            check = 1;
        end

        if(check == 0 && vector(j) == '0')
            lzc = lzc + 1;
        end
    end

    % Make initial guess y0
    B = w_bits - f_bits - lzc - 1;

    if(mod(B,2) == 0)
        even = 1;
    else
        even = 0;
    end

    if(even == 1)
        A = 1.5*B;
    end
end

```

```

else
    A = 1.5*B + 0.5;
end

Xa = bitsrl(a(i),A);
Xb = bitsrl(a(i),B);
Xb_dec = ufi(Xb);

% Use 7-bit addressing to match format of LUT in simulation
Xb = fi(Xb_dec.data,0,8,7);

% Convert from fixed point to decimal for -3/2 power exponent
Xb_dec = ufi(Xb);
Xb_data = Xb_dec.data ^ -1.5;
LUT = fi(Xb_data,0,w_bits,f_bits);

if(even ==1)
    y0 = fi(Xa * LUT,0,w_bits,f_bits);
else
    y0 = fi(Xa * LUT*0.7071067812,0,f_bits,f_bits);
end

y0 = fi(y0,0,w_bits,f_bits);

% Update y0 using Newton's iterations
for k = 1:N_iterations
    input_y = y0;

    y0 = fi(a(i)*y0*y0,0,w_bits,f_bits);
    y0 = fi(3-y0,0,w_bits,f_bits);
    y0 = fi(y0*input_y*0.5,0,w_bits,f_bits);
end

testbench = [testbench fi(y0,0,w_bits,f_bits)];

end

```

Read from ModelSIM output file

```

fileID = fopen('output.txt','r');

a = []

for i = 1:100

    float = 0;
    int = 0;
    frac = 0;

```

```

line_a = fgetl(fileID);

% Calculate integer component
for i = 1:diff
    int = int + str2num(line_a(i))*2^(diff-i);
end

% Calculate fractional component
for i = diff+1:w_bits
    frac = frac + str2num(line_a(i+1))/(2^(i-diff));
end

float = int + frac;
fixed = fi(float,0,w_bits,f_bits);

a = [a fixed];

end

ModelSim = a;
MATLAB = testbench;

fclose(fileID);

a =

    []

```

Verification

```

Inputs_verified = 0;

% Compare results between ModelSim output and MATLAB emulation
for i = 1:100
    fprintf('\n')
    disp("Input: " + i + "
-----")
    ModelSim_Result = bin(ModelSim(i))
    MATLAB_Result = bin(MATLAB(i))

    check = 0;
    for j = 1:w_bits
        if(ModelSim_Result(j) ~= MATLAB_Result(j))
            check = 1;
            disp('Verification failure in bit ')
            disp(w_bits-j)
        end
    end

    if(check == 0)

```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Input: 100 -----

```
ModelSim_Result =
```

```
'00000000000000000000000000101100001'
```

MATLAB_Result =

[illegible]

Verification success!

Inputs_verified =

100

Published with MATLAB® R2019b