Brief Introduction on ReMapping Method

ver 1.0.1

This document briefly describes how the ReMapping method, an optimization for the range mapping method in Mitchell's logarithmic approximation method, is composed.

ReMapping

The remapping method is derived directly from the range mapping method. In short, this method separately and directly remaps the mantissa m1 to each result segment line of range mapping method, rather than through a process of range mapping followed by error correction.

According to the range mapping method, domain of the Mitchell curve is uniformly divided into 4 regions by Gr, most of which has 3 segments divided by m2 except the 4-segment first region.

For segment 2 to 4 in each region, they are in the form of

$$result = m_2 + f_i = G_r m_1 + G_r + f_i - 1(*)$$

As Gr and fi are constants, the result in segment 2 to 4 is a linearly dependent on m1. Therefore, after finding at least two points on each segment line of the range mapping method, we can easily find out the coefficients which determines how we should map m1 directly to the line.

For segment 1, things are different. To minimize the error, an additional term of 2^{-3}*$m2_{MSB7}$ is included in the (*) expression, resulting in a stepwise function of 12 pieces.

As of now, our solution to this problem is to find as many points as possible and fit these points to a line and use this line as the applicable remapping object.

The reason is that one of the goals is to achieve best FoM possible, including circuit area and timing. This asks for uniform (or at least, very similar) structure for each segment, so that circuit operation can be completed by one single set of datapath. Also, this solution provides better error distribution due to its smoothness.

After fitting 1000 points, we have a single line as follows with SSE = 7.893e-05 and R-square = 0.9999 regarding to the original points, taking segment 1 in region 1 as the example.

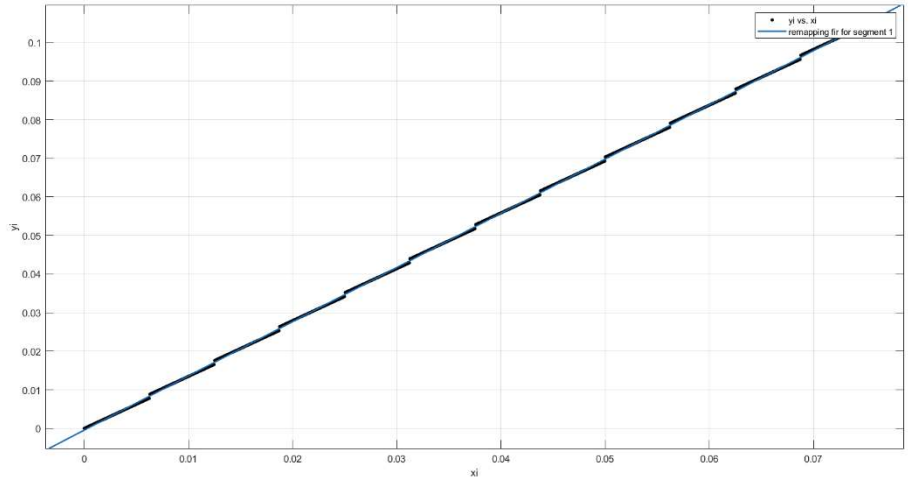

Figure 1. original segment 1 in region 1 and the fitted line

Now that we have coefficients for all segments, it's easy to remap m1 to the object lines by applying these sets of coefficients on m1 in accordance with the original segments.

Implementation

The introduction will be in file sequence: castrARM4.m, FindPoints.m, Fitting.m, FindCoefficients.m, AlgoReMapping_16bit.m, CastrARM16.m, FindPoints2.m. The underlined is the body of this method.

castrARM4.m

This function is a reduced version of the original function, AlgoRangeMapping4.m, with modifications of changing the inputs from a number to be calculated into only the mantissa *m1* to be approximated (that is, getting rid of the characteristic *k)*, and forcibly including *m1*=1 into the range.

The goal of this function reduction is to ease the calculation of specific points corresponding to different *m1* values.

FindPoints.m

This script uses the castrARM4 function to calculate *m1*, *m2*, and the range mapped *gm1* for all critical points for reference purpose.

```
m2 = [
    0.25, 0.34375, 0.375, 0.53125, 0.5625;
    0.25, 0.34375, 0.375, 0.5, 0.5;
    0.3125, 0.34375, 0.375, 0.53125, 0.53125;
    0.3125, 0.34375, 0.375, 0.5, 0.5; %forcibly included m1 = 1 into the orginal algo
    ];

    m1 = (m2+1)./Gr-1;
    gm1 = reshape(arrayfun(@castrARM4, m1(:)), 4, 5);
```

Figure 2. part of the codes in FindPoints.m

The results are written into FindPoints.txt file.

Fitting.m

From now on, we are getting into the ReMapping method. Fitting function can be invoked in such a form:

$$poly = Fitting(rgn, seg)$$

where *rgn* and *seg* are the number for region and segment respectively, and poly is the polynomial coefficients.

This function fits a reconfigurable number (e.g., 1000) of points in a certain segment and returns its polynomial coefficients.

```
    point_num = 1000; %number of points to be fitted

    m1 = (m2+1)./Gr-1;
    step = (m1(rgn,seg+1)-m1(rgn,seg))/point_num;
    xi = (m1(rgn,seg) : step : m1(rgn,seg+1)-step);
    yi = arrayfun(@castrARM4, xi);

    poly = polyfit(xi, yi, 1);
```

Figure 3. part of the codes in Fitting.m

FindCoefficients.m

This script uses two nesting loops of Fitting function to acquire all sets of coefficients for all segments and outputs them into FindCoefficients.txt file.

```
for rgn = (1 : 4) %region 1
    You, 4 minutes ago | 1 author (You)
    if rgn == 1
        You, 4 minutes ago | 1 author (You)
        for seg = (1 : 4)
            poly = Fitting(rgn, seg);
            fprintf(fid, '%f ', poly);
            fprintf(fid, '\n');
            %xi = m1(rgn, seg) : 0.00001 : m1(rgn, seg+1)-2^-16;
            %yi = polyval(poly, xi);
            %plot(xi, yi);
            %hold on;
        end
    You, 4 minutes ago | 1 author (You)
    else
        You, 4 minutes ago | 1 author (You)
        for seg = (1 : 3)
          poly = Fitting(rgn, seg);
            fprintf(fid, '%f ', poly);
            fprintf(fid, '\n');
            %xi = m1(rgn, seg) : 0.00001 : m1(rgn, seg+1)-2^-16;
            %yi = polyval(poly, xi);
            %plot(xi, yi);
            %hold on;
        end
    end
end
```

Figure 4. part of the codes in FindCoefficients.m

Also, an overview plot of current method can be given by uncommenting.

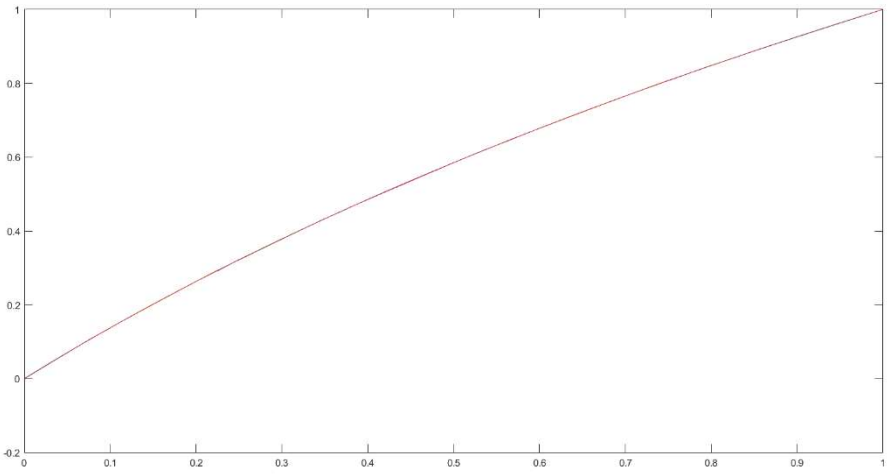

Figure 5. ReMapped curve and log(m1+1) on the same plot

AlgoReMapping_16bit.m

With the coefficients acquired, it's rather simple a logic to complete the ReMapping method using a set of conditionals as following:

```
function rslt = AlgoReMapping_16bit(num)
%ALGOREMAPPING opmitised method for range mapping
%   After figuring out coefficients by fitting, we apply them to the
%   remapping method. This function calculates one result each time.

format float;

fid = fopen('FindCoefficients.txt', 'r');
coef = fscanf(fid, '%f', [2, 13]);

[k, m] = KeyValues(num);

You, 2 days ago | 1 author (You)
if (0 <= m && m < 0.075)
    m_r = coef(1, 1)*m + coef(2, 1);
You, 2 days ago | 1 author (You)
elseif(0.075 <= m && m < 0.1)
    m_r = coef(1, 2)*m + coef(2, 2);
You, 2 days ago | 1 author (You)
elseif(0.1 <= m && m < 0.225)
    m_r = coef(1, 3)*m + coef(2, 3);
```

```
elseif(5/6 <= m && m < 1)
    m_r = coef(1, 13)*m + coef(2, 13);
end

fclose('all');

rslt = m_r + k;

end
```

Figure 6. part of the codes in AlgoReMapping_16bit.m

CastrARM16.m and FindPoints2.m

Like what has been done in castrARM4.m and FindPoints.m, we can calculate some of the method results and compare them. (All rounded to six decimal places)

m1	ReMapping	difference with range mapping ($*10^{-4}$)	difference with Log2(m+1) ($*10^{-4}$)
0	-0. 000447	-4.470	-4.47
0.075	0.104941	-5.28	6.04
0.1	0.138184	0	6.80
0.225	0.291504	0	12.78
0.25	0.321332	-4.45	5.96
0.34375	0.427246	0	9.81
0.375	0.459961	0	5.29
0.5	0.584596	-3.65	-3.67
15/28	0.619629	0	7.19
4/7	0.652832	0	7.55
0.75	0.806764	-3.65	-5.91
19/24	0.842285	0	9.83
5/6	0.873535	-14.6	-9.35
1	1	0	0

Conclusion

Though further simulation is still to be done, this first version of ReMapping method can be believed as promising. The comparison of critical points results indicates close (or even better) accuracy with the original range mapping method. Following is a 16-bit simulation result comparison of the current method versus the range mapping method.

```
RangeMapping          : Min Error      = -0.0028911713, Input    =     29867
                      : Max Error      = +0.0022166622, Input    =     28087
                      : Min Error(%)   = -0.0565948883, Input    =        25
                      : Max Error(%)   = +0.0356808541, Input    =        55

AlgoReMapping_16bit (16-bit) : Min Error      = -0.0017217504, Input    =     57343   better
                      : Max Error      = +0.0012850311, Input    =     40141     better
                      : Min Error(%)   = -0.0230611567, Input    =        39     better
                      : Max Error(%)   = +0.0457763672, Input    =         2     worse
```

Figure 7. 16-bit simulation results comparison versus the original method

However, due to the non-linearity of segment 1 in each region, some of the results are unreliable especially those near the origin, as can be told by a larger maximum relative error's appearance on 2. This can be improved significantly by introducing several more segments with a length of LSB in each segment 1 considering both the logarithmic curve and the range mapping curve. The coefficients and number of these segments can be determined by simulation, which is exactly the next one on the to-do list.