5LIK0: Embedded Signal Processing Systems, Assignment 2: Fixed Point Design

Marc Geilen (m.c.w.geilen@tue.nl)

May 15, 2024

Important Notes:

- This assignment may be executed together with one other student, or individually (if you do it individually you have to do all the work alone).
- Any form of collaboration on the assignments with other students than your partner is not allowed.
- The assignment assumes that you are familiar with the lectures on Fixed Point Design and Section 6, *Numerical Representations of Signals* of the lecture notes.

The deadline for submission of Assignment 2 is Friday June 14 2024, 23.59h. It should be submitted on Canvas as a single zip file named `assignment2_<id1>_<id2>.zip`, where `<id1>` and `<id2>` are your respective student ID numbers. The zip archive should include the solution MATLAB scripts, i.e., at least `testCordic_sqrt.m`, in which optimal representations are used, and your new script to support a wider range, and a short report (2–4 pages, excluding figures or tables could suffice) in pdf format describing your analysis and design choices leading to your final solution.

In this assignment you will perform a fixed-point *analysis* and *design* on a small function. You will consider the following MATLAB function that uses the so-called CORDIC operation to compute the mathematical *square root* function. We discuss the CORDIC rotation function in the course lectures, but for the assignment it is not important how it works exactly. More information can be found here: `https://www.mathworks.com/help/fixedpoint/examples/compute-square-root-using-cordic.html`

The function has one argument, `v`, and one parameter, `N`. The function computes from the input `v`, which should be **in the range** $[0.5, 2)$, an approximation of $\sqrt{\mathtt{v}}$. The CORDIC function uses only the arithmetic operations of addition and multiplication (in fact, only one multiplication, with a constant). The CORDIC function can therefore be efficiently realized in hardware.

The following listing[1] shows its implementation in MATLAB. The MATLAB script is provided with the assignment on Canvas.

[1] adapted from `https://www.mathworks.com/help/fixedpoint/examples/compute-square-root-using-cordic.html`

```
% Use the cordic to compute the sqrt operation
function r = sqrt_cordic(v, N)
% v a number in the range [0.5, 2)
% compute for N iterations

% set the initial values for x and y
x = v + 0.25; y = v - 0.25;

K = 4; % to perform an extra iteration when k==K
for k = 1 : N
    %-----------------------------------------------------
    if y < 0
        s = 1;
    else
        s = -1;
    end
    xnew = x + s*bitsra(y, k);
    ynew = y + s*bitsra(x, k);
    x = xnew;
    y = ynew;
    %-----------------------------------------------------
    % (repeat the loop once more for k=4, 13, 40, 121, ...)
    if k == K
        K = 3*K + 1;  % compute the next time to repeat
        % the following lines are identical to the part
        % between horizontal lines above
        %-----------------------------------------------------
        if y < 0
            s = 1;
        else
            s = -1;
        end
        xnew = x + s*bitsra(y, k);
        ynew = y + s*bitsra(x, k);
        x = xnew;
        y = ynew;
        %-----------------------------------------------------
    end
end
inv_Gain = 1.207497067763072;  % inverse of CORDIC gain for large N
r = x * inv_Gain;
end
```

The function additionally has a parameter `N`. The CORDIC function is iterative (corresponding to the `for` loop on Line 10 in the code). `N` determines the number of iterations that are computed. More iterations provide a more accurate result. (Note, however, that the number of iterations is not *equal* to `N`.)

Note that the `bitsra(x, k)` MATLAB function computes a bit-shift operation of the number `x` by `k` bits to the right. This is equivalent to a multiplication with 2^{-k} and hence `+ s*bitsra(x, k)` is an addition or subtraction of $x \cdot 2^{-k}$, depending on the value of `s`.

You may assume that the input data, `v`, on which the function is going to be applied have a random, uniform distribution over the range $[0.5, 2)$.

Address in your report and scripts answers to the following questions.

Question 1 ARITHMETIC OPERATIONS

Determine the number and type of arithmetic operations that are performed by the `sqrt_cordic` function, *in a hardware implementation of it*, to compute the square root in relation to the parameter `N`. Indicate which type of operations are applied (e.g., division, multiplication, addition) and on what type of data (integers or fractional data, it does not matter here whether the fractional data is in floating point or fixed point).

Question 2 FIXED POINT DESIGN

In the archive provided with the assignment on Canvas, you find three MATLAB scripts.

- `sqrt_cordic.m` contains the function as shown above. It does all its computations in double precision floating point representation. It should be taken as a reference behavior of the CORDIC function to compare a fixed point implementation against.
- `sqrt_cordic_fixpt.m` is a version of the same function that computes the result with fixed point representations and computations. The full script is reproduced at the end of this assignment description.
- `testCordic_sqrt.m` is a script that you can use, **and adapt, extend**, if needed, to evaluate the accuracy of the computations in the fixed point function, as compared to the floating point CORDIC computation and MATLAB's `sqrt` function.

The goal of the assignment is to perform a fixed-point design of the `sqrt_cordic` function, including the required number of iterations, i.e., to decide on the best possible fixed point representations for the variables and the computations in the `sqrt_cordic_fixpt` function so as to satisfy the following *constraint on the expected error in the computed output*, given the distribution of input values specified above.

$$E(|r_{float} - r_{fixpt}|^2) < 3 \cdot 10^{-5}$$

in this equation, r_{float} is the square root computed according to double precision floating point computation with MATLAB's `sqrt` function and r_{fixpt} is the square root computed by the fixed-point CORDIC function (`sqrt_cordic_fixpt.m`).

The objective is further to *minimize the latency* (i.e., the number of iterations in the `cordic` function). If the objectives turn out to be conflicting, then find a *balanced compromise*.

Recall that errors are introduced by fixed-point representations, in general, by rounding error to the representations and by saturation in case of overflows of the range. It may be necessary to strike a trade-off between the errors introduced by both effects.

It is indicated in the scripts which parameters you may vary and which parameters cannot be changed.

Your choice of solutions may be based on analytical reasoning about the data and the computations, or on a structured experimental evaluation, or both. **The optimal result needs to be explained, so exhaustive testing of all possible parameter values without justification is not sufficient.**

Question 3 RANGE

The CORDIC square root function only works for input values in the range $v \in [0.5, 2)$. This range is often too small for practical use. Show how the `sqrt_cordic_fixpt` function can be used to compute square roots for a wider range of input values. Hint: see again the page: `https://www.mathworks.com/help/fixedpoint/examples/compute-square-root-using-cordic.html`. Write a MATLAB function, called `sqrt_cordic_widerange_fixpt`, that works with input data of the same fixed-point data type that you have determined in the previous question and computes an approximation of the square root operation using the `sqrt_cordic_fixpt` function. Comment on the accuracy of the function across the range of input values it supports.

Appendix Matlab Code

Listing: `sqrt_cordic.m`

```
% Use the cordic to compute the sqrt operation
function r = sqrt_cordic(v, N)
% v a number in the range [0.5, 2)
% compute for N iterations

% set the initial values for x and y
x = v + 0.25; y = v - 0.25;

K = 4; % to perform an extra iteration when k==K
for k = 1 : N
    %-----------------------------------------------------
    if y < 0
        s = 1;
    else
        s = -1;
    end
    xnew = x + s*bitsra(y, k);
    ynew = y + s*bitsra(x, k);
    x = xnew;
    y = ynew;
    %-----------------------------------------------------
    % (repeat the loop once more for k=4, 13, 40, 121, ...)
    if k == K
        K = 3*K + 1;  % compute the next time to repeat
        % the following lines are identical to the part
        % between horizontal lines above
        %-------------------------------------------------
        if y < 0
            s = 1;
        else
            s = -1;
        end
        xnew = x + s*bitsra(y, k);
        ynew = y + s*bitsra(x, k);
        x = xnew;
        y = ynew;
        %-------------------------------------------------
    end
end
inv_Gain = 1.207497067763072;  % inverse of CORDIC gain for large N
r = x * inv_Gain;
end
```

Listing: `sqrt_cordic_fixpt.m`

```
% Use the cordic to compute the sqrt operation
% fixed point version
% ThetaLUTFP contains the fixed point lookup table
% DAT_BW, DAT_FL, DAT_S, F are the fixed point parameters,
% see testCordic_sqrt.m
function r = sqrt_cordic(v, N, DAT_BW, DAT_FL, DAT_S, F)
% compute for N iterations
% First: tune the inputs
c = 0.25; % A constant
x = v + c; y = v - c; % add input and the constant to x an y
K = 4; % to repeat k every 3K+1
for k = 1 : N
    %-------------------------------------------------
    % Compare this code segment with that in cordic.m
    if y < 0
        s = 1;
    else
        s = -1;
    end
    xnew = x + s*bitsra(y, k);
    ynew = y + s*bitsra(x, k);
    x = fi(xnew, DAT_S, DAT_BW, DAT_FL, F);
    y = fi(ynew, DAT_S, DAT_BW, DAT_FL, F);
    %-------------------------------------------------
    % (repeat the loop once more for k=4, 13, 40, 121, ...)
    if k == K
        if y < 0
            s = 1;
        else
            s = -1;
        end
        xnew = x + s*bitsra(y, k);
        ynew = y + s*bitsra(x, k);
        x = fi(xnew, DAT_S, DAT_BW, DAT_FL, F);
        y = fi(ynew, DAT_S, DAT_BW, DAT_FL, F);
        K = 3*K + 1;  % compute the next time to repeat
    end
    %-------------------------------------------------
end
inv_Gain = fi(1.207496866840026, DAT_S, DAT_BW, DAT_FL, F);
r = x * inv_Gain;
end
```

Listing: `testCordic_sqrt.m`

```
% testCordic_sqrt.m
%
% the following parameters define the fixed point representation of the
% data variables in the computation of the square root
DAT_BW = 20;    % the bit width of the data variables
DAT_FL = 5;     % the fraction length of the data variables
DAT_S = true;   % the signedness of the data variables

% the number of iterations of the CORDIC algorithm
NIter = 5;

% do not change any of the fixed point parameters below this point!
% default parameters for fixed point arithmetic
F = fimath('OverflowAction','Saturate',...
    'ProductMode','SpecifyPrecision',...
    'ProductWordLength', DAT_BW,...
    'ProductFractionLength', DAT_FL,...
    'SumMode', 'SpecifyPrecision',...
    'SumWordLength', DAT_BW,...
    'SumFractionLength', DAT_FL,...
    'CastBeforeSum', true);

step  = 2^-6;

% input values in the range [.5, 2)
v = 0.5:step:(2-step);
% fixed-point inputs in range [.5, 2)
v_fixpt = fi(v, DAT_S, DAT_BW);

x_sqr = zeros(1,length(v)); % An array to hold the floating point cordic values
x_sqr_fixpt = zeros(1,length(v)); % An array to hold the fixed point cordic values

% Get the CORDIC outputs for comparison
% and plot the error between the MATLAB reference and CORDIC sqrt values
for i=1:length(v)
    x_sqr(i) = sqrt_cordic(v(i), NIter);
    x_sqr_fixpt(i) = sqrt_cordic_fixpt(v_fixpt(i), NIter, DAT_BW, DAT_FL, DAT_S, F);
end

% compute the reference floating-point results with Matlab sqrt function
x_ref = sqrt(v);

% Make a plot of the results
figure;
subplot(2,1,1); hold on;
plot(v, double(x_sqr), 'r-');       % plot the floating point CORDIC
plot(v, double(x_sqr_fixpt), 'g-'); % plot the fixed point CORDIC
plot(v, x_ref, 'b-');               % plot the reference
legend('CORDIC float', 'CORDIC fixpt', 'Reference', 'Location', 'SouthEast');
title('CORDIC Square Root, CORDIC Fixed Point (In-Range) and MATLAB Reference Results');

subplot(2,1,2); hold on;
absErr_fltpt = abs(x_ref - double(x_sqr));       % compute error in the floating point CORDIC
absErr_fixpt = abs(x_ref - double(x_sqr_fixpt)); % compute error in the floating point CORDIC
plot(v, absErr_fltpt, 'r-');
plot(v, absErr_fixpt, 'g-');
title('Absolute Error (vs. MATLAB SQRT Reference Results)');
```