## 1 Introduction

All fractional-N synthesizers have frequencies at which spurious products are generated; this is also the case with the CMX940 High Performance Synthesizer. A discussion of these spurious products and methods to avoid them can be found in Section 7.4, “Spurs and Spur Avoidance” in the CMX940 Datasheet This Application Note has been written to explain how such problematic spurs may be avoided by using a Spur Avoidance Algorithm (SAA). Using a worked example, the algorithm is demonstrated and shown to achieve good results.

The SAA was developed in Octave / Matlab. A version of the Octave code with explanatory comments may also be found in an Annex of this document.

Additionally, a ‘C’ code implementation for the PE0003 Universal Interface Card and EV9400 CMX940 Evaluation Kit may be found on the CML website on the CMX940 product page.

## 2 History

| Version | Changes | Date |
|---|---|---|
| 1.0 | First published | April 2022 |
| | | |

# 3 The Spur Avoidance Algorithm

## 3.1 Assumptions

This document assumes that the reader is already familiar with the CMX940.

The present implementation of the SAA assumes:

- The user is using a CMX940 and EV9400 Evaluation Board in default build configuration, specifically:
  - the RVCO (L6 on EV9400) inductor is 4.7nH
  - the reference oscillator (MCLK) is 38.4MHz.
- The EV9400 is connected to a PE0003 Universal Interface Card.

## 3.2 The Algorithm

The SAA launches from the command window in Octave in this example to achieve 400MHz (f) output frequency, “SpurCalcC(400)” is written. The SAA then calculates and returns a set of C-BUS register values that should provide the best spur performance for the target output frequency.

Provided here is a summary of the SAA process:

**1)** The REF_PLL has a set of multipliers / divider (M and P) values that are configured to set the comparison frequency for the SPLL. The M and P references refer to the following registers:
M = REF_PLL_MDIV ($7A) and
P = SYN_SDIV ($4D) x SPLL_RDIV ($6A), where the SPLL_RDIV value = 1.

**2)** Based on the previous assumptions above, valid M and P values are as follows (where each value is in the form [M,P]:

[25,8], [25,9], [26,9], [27,9], [27,10].
or
[27,9], [27,10], [26,9], [25,8], [25,9].

When the SAA is looking for spurs it will systematically run through either of these sets of values and as soon as it find an M,P pair without a spur it will finish.

To improve performance, there are two orders that this set of values can take.
The first set of values is chosen when 499.2 MHz ≤ f < 729.6 MHz or f ≥ 998.4 MHz.
The second set of values is chosen when f < 499.2 MHz or 729.6 MHz ≤ f < 998.4 MHz.

**3)** Next the SAA picks an output divider ODIV (REF_LO_DIV ($7B)) value for which it ensures that the operating frequency of the SVCO is in the range of 2720 – 4080 MHz. The potential divider values are as follows: 1, 2, 3, 4, 6, 8, 12, 16, 20, 28, 40, 56.

Note that a ÷1 results in reduced output power, see the CMX940 Datasheet

**4)** Now we analyse the potential spurs (in the VCO range), of which there are three that the SAA considers:

a. **MCLK Spurs**: these spurs occur at integer multiples of MCLK*M/P MHz.
b. **RVCO Spurs**: these spurs occur at half-integer multiples of MCLK*M MHz (the RVCO frequency).

c. **Integer Boundary Spurs**: these spurs occur at divisions of the N value (or SVCO / Fcomp). There are many types of boundary spurs, and the SAA considers all these values up to and including fifth order spurs.
For example an xth order will appear when the frequency $\frac{y*N}{x} + k * N$ occurs, where y ϵ {1,2,…, x - 1} and k ϵ N. It should also be noted that if we are looking at a VCO frequency of fVCO and a boundary spur frequency of fD of order x, then the offset frequency of the boundary spur from the VCO frequency will be x*( fVCO - fD).

Now that we are able to calculate different spur types, the SAA runs through the different M,P values and divider settings. If a combination of settings is found that does not create a spur within 5.5 MHz, it will finish its search for suitable values. If however spurs are present for all possible settings within this range, the SAA will run back through again but allowing fifth order spurs to occur provided they are outside the 550 kHz limit. This is acceptable as fifth order spurs are typically of very low amplitude. If this second attempt fails to provide a suitable setting, the SAA tracks what the 'nearest spur' is for each potential combination and will choose the values that give the furthest spur offset.

Note that the 5.5 MHz and 550 kHz window spur thresholds are variables based on the default ~100 kHz loop filter bandwidth settings of the EV9400 and chosen for typical 400 – 470 MHz applications. These window spur thresholds could be modified if different loop bandwidths are being used.

**5)** Finally the SAA calculates frequency and division values and converts these into the hex values for REF_PLL input, REF_PLL_MDIV, SYN_SDIV , N (SPLL_IDIV - integer part and SPLL_FDIV0/1 - fractional part) and these are written to the device.

The Octave / Matlab algorithm that has been developed and its accompanying explanatory comments can be found in annex A of this document.

### 3.3 Results

Typical results are shown in Figure 1 (spur is a (2/3) x $f_{comp}$ product) and Figure 2 with optimised settings calculated by the SAA.

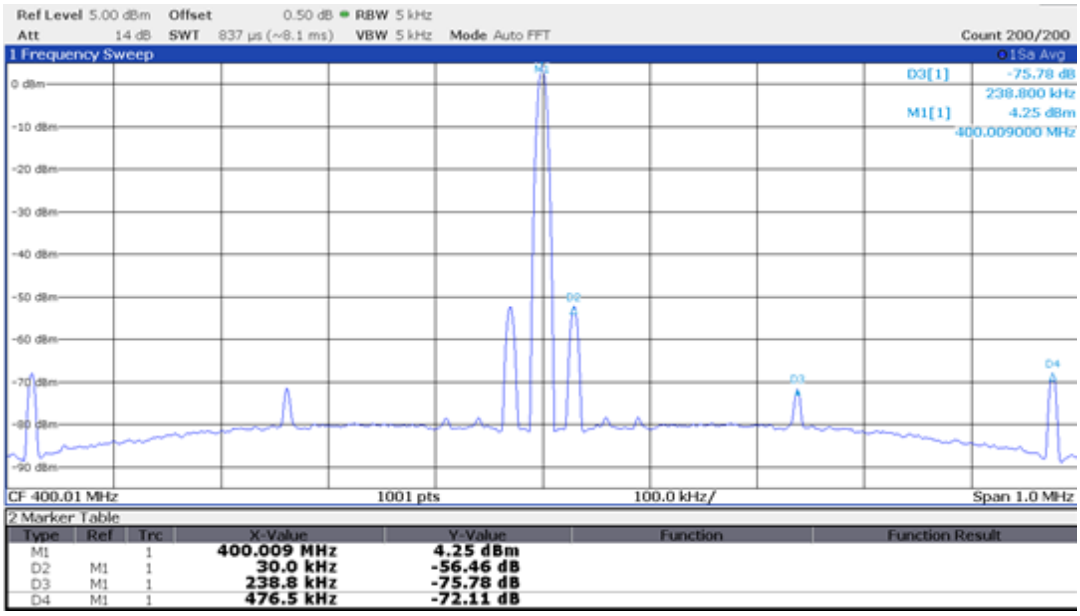


**Figure 1 – Spectrum with default settings (REF_PLL = 960 MHz, $f_{comp}$ = 120 MHz)**

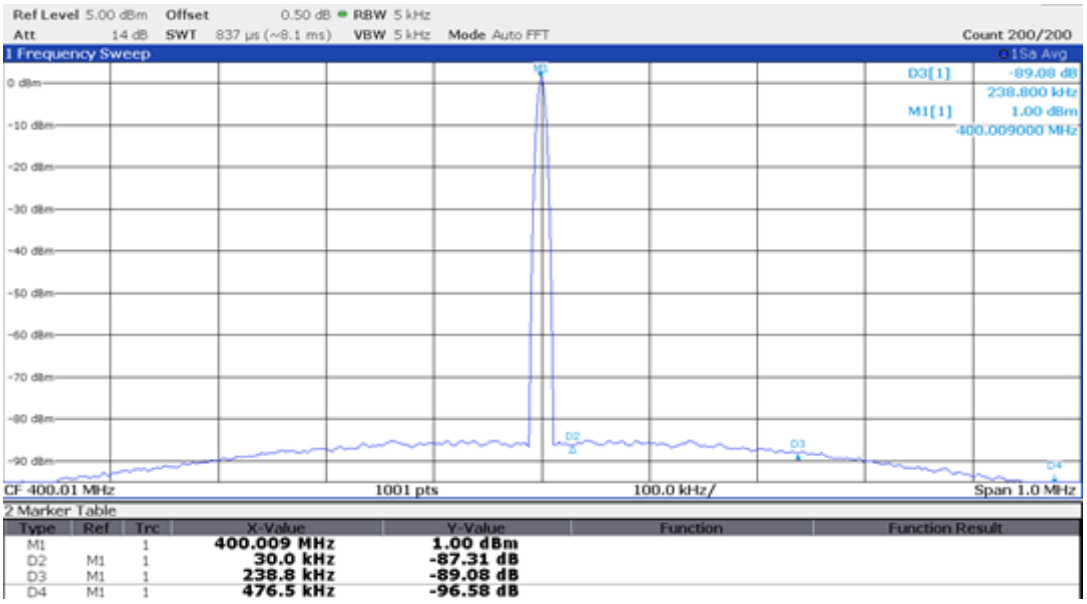


**Figure 2 – Spectrum with settings selected by SAA software [27,9] (REF_PLL = 1036.8 MHz, $f_{comp}$ = 115.2 MHz)**

# 4 Conclusion

The SAA has been demonstrated to be an effective way to mitigate spurious products in the CMX940. The Octave / Matlab algorithm and a 'C' code version is available from the CMX940 product page on the CML website and may be freely modified for each target application as necessary.

# Annex A – Octave / Matlab Implementation

```
% Published Octave Script
% March 2022

function retval = SpurCalcC(freq)
% New to replace RefVal
  function retval = RefVal(freq)
    if freq < 499.2
      retval = [27,9;27,10;26,9;25,8;25,9];
    elseif freq < 729.6
      retval = [25,8;25,9;26,9;27,9;27,10];
    elseif freq < 998.4
      retval = [27,9;27,10;26,9;25,8;25,9];
    else
      retval = [25,8;25,9;26,9;27,9;27,10];
    endif
  endfunction
% Eend


function retval = Divider(freq)
% This function takes all possible divider values (j = ...) and creates a vector which contains all ODIV
divider values that are valid for an SVCO operating range of 2720 - 4080 MHz
    retval = [];
    for j = [1,2,3,4,6,8,12,16,20,28,40,56]
      VCO = freq*j;
      if VCO >= 2720 && VCO <= 4080
        retval = [retval ; [j]];
      endif
    endfor
  endfunction


function retval = ClockSpur(freq,M,P)
% This function calculates the values of any clock spur within 1 MHz of the output frequency
    Clock = 38.4*M/P;
    if mod(freq+1,Clock) <= 2 % This condition determines whether there are any clock spurs within 1
MHz of the output frequency
    retval = abs(mod(freq+1,Clock)-1)*1000; % This value tells us the distance the clock spur sits from
the output frequency
    else
    retval = 10^6; % We choose this value if there are no clock spurs within 1 MHz of the output
frequency. This value is suitably large such that the SAA will not consider there to be a problematic
spur here
    endif
  endfunction


function retval = VCOSpur(freq,M,P)
    RVCO = 38.4*M/2;
    SVCO = freq*4;
 if mod(SVCO+4,RVCO) <= 8 % This condition determines whether there are any VCO (or half VCO)
spurs within 4 MHz of the output frequency
      retval = abs(mod(SVCO+4,RVCO)-4)*250;   % This value tells us the distance the VCO (half
VCO) spur sits from the output frequency
 else
```

```
    retval = 10^6; % We choose this value if there are no VCO (or half VCO) spurs within 4 MHz of the output frequency. This value is suitably large such that the SAA will not consider there to be a problematic spur here
   endif
  endfunction


function retval = BoundarySpur(freq,M,P,OD,test)
% New argument 'test' is introduced
% This function calculates the position of any boundary spurs within various specified ranges of the output frequency
   freq = freq*OD; % Calculating SVCO frequency
   SpurFrac = [1 1/2 1/3 1/3 1/4 1/4 1/5 1/5 1/5 1/5]; % The boundary spur types we will be looking out
   Fref = 38.4*M/P;
   Margin = 2.5/Fref;
   G1 = freq; % SVCO frequency - for later reference
   freq0 = freq/Fref; % Calculating the divider value, N
   freq1 = floor(freq0 + 0.5); % Calculates the nearest integer to freq0 (N)
   freq = freq0 - freq1; % Calculates position of freq0 from nearest integer
   H = [10^6,10^6,10^6,10^6,10^6,10^6,10^6,10^6,10^6,10^6]; % This line makes the default values of all boundary spurs large enough that the SAA will not consider them to be initially problematic
% Test for fifth vs Integer Spurs
   FifthSpur = 1; % This allows the range over which the fifth spur is considered to be reduced in order to find a suitable solution
   if test == 1
     FifthSpur = 10;
   endif
%END: LOOK AT TEST FOR NEW VALUES AT THE END OF THE VECTOR: '5' replaced by '(5*FifthSpur)' Also integer spur margin is increased by factor '*2.2'

   Test = [logical(abs(freq) <= Margin*2.2) logical(abs(0.5 - mod(freq0,1)) <= Margin/2) logical(abs(1/3 - (freq)) <= Margin/3) logical(abs(-1/3 - (freq)) <= Margin/3) logical(abs(0.25 - (freq)) <= Margin/4) logical(abs(-0.25 - (freq)) <= Margin/4) logical(abs(0.2 - (freq)) <= Margin/(5*FifthSpur)) logical(abs(0.4 - (freq)) <= Margin/(5*FifthSpur)) logical(abs(-0.4 - (freq)) <= Margin/(5*FifthSpur)) logical(abs(-0.2 - (freq)) <= Margin/(5*FifthSpur))]; % This provides a vector of ones and zeros denoting whether boundary spurs are present within the specified ranges
%END
%  G calculates the position of all boundary spurs that we are considering
   G(1) = G1/OD;
   G(2) = G1;
   G(3) = abs(-Test(1)*Fref*freq);
   G(4) = abs(Test(2)*Fref*(0.5-mod(freq0,1)));
   G(5) = abs(Test(3)*Fref*(1/3-(freq)));
   G(6) = abs(Test(4)*Fref*(-1/3-(freq)));
   G(7) = abs(Test(5)*Fref*(0.25-(freq)));
   G(8) = abs(Test(6)*Fref*(-0.25-(freq)));
   G(9) = abs(Test(7)*Fref*(0.2-(freq)));
   G(10) = abs(Test(8)*Fref*(0.4-(freq)));
   G(11) = abs(Test(9)*Fref*(-0.4-(freq)));
   G(12) = abs(Test(10)*Fref*(-0.2-(freq)));
   if Test(1) != 0 || Test(2) != 0 || Test(3) != 0 || Test(4) != 0 || Test(5) != 0 || Test(6) != 0 || Test(7) != 0 || Test(8) != 0 || Test(9) != 0 || Test(10) != 0
 %  H considers the test vector and if a spur is problematic (i.e. Test = 1) then the spur location is kept and if it is not then the position is increased so that the SAA considers it not a problem
     H(1) = G(3) + logical(Test(1) == 0)*10^6;
     H(2) = G(4)*2 + logical(Test(2) == 0)*10^6;
     H(3) = G(5)*3 + logical(Test(3) == 0)*10^6;
     H(4) = G(6)*3 + logical(Test(4) == 0)*10^6;
     H(5) = G(7)*4 + logical(Test(5) == 0)*10^6;
     H(6) = G(8)*4 + logical(Test(6) == 0)*10^6;
```

```
    H(7) = G(9)*5 + logical(Test(7) == 0)*10^6;
    H(8) = G(10)*5 + logical(Test(8) == 0)*10^6;
    H(9) = G(11)*5 + logical(Test(9) == 0)*10^6;
    H(10) = G(12)*5 + logical(Test(10) == 0)*10^6;
  endif

  I = min(H(1:10))*1000;
  retval = I; % This returns the smallest boundary spur value
 endfunction

function retval = MPcomb(freq,RefVal,OD)
%SpurLoc = -1;
  for k = 1:size(OD,1) % Searches through divider settings
    for j = 1:size(RefVal,1) % Searches through [M,P] settings

      M = RefVal(j,1);
      P = RefVal(j,2);
      %  Calculates the smallest spur for specific settings
      Spur = min([BoundarySpur(freq,M,P,OD(k),0),VCOSpur(freq,M,P),ClockSpur(freq,M,P)]);  %
Boundary spur now has an additional argument (4th one)

      if (Spur > 5500) % New increase spur search size
        % returns values if spur is not within 5.5 MHz
        retval = [freq,M,P,OD(k)];
        return;
      endif

    endfor
  endfor

  % if no immediate match then retest for 5th spur
  % ..and keep track of best result

% Running the same test again but with less fifth spur margin
% Elseif k == size(OD,1) && j == size(RefVal,1)
   SpurLoc = -1;
   for k = 1:size(OD,1)
     for j = 1:size(RefVal,1)

        M = RefVal(j,1);
        P = RefVal(j,2);

        Spur = min([BoundarySpur(freq,M,P,OD(k),1),VCOSpur(freq,M,P),ClockSpur(freq,M,P)]);
%Boundary Spur now has an additional argumant (4th)

        if (Spur > 5500)
           retval = [freq,M,P,OD(k)];
           return;
        elseif Spur > SpurLoc
           SpurLoc = Spur;
           retval = [freq,M,P,OD(k)];
        endif

      endfor
   endfor
% End of test

 endfunction
```

```
function retval = int(x)
  retval = x - mod(x,1);
endfunction


function retval = Frequencies(input)
  MCLK = 38.4;
  Out = input(1);
  M = input(2);
  P = input(3);
  OD = input(4);
  RVCO = MCLK*M;
  Comp = RVCO/P;
  SVCO = Out*OD;
  N = SVCO/Comp;
  retval = [MCLK,M,RVCO,P,Comp,N,SVCO,OD,Out];
endfunction


function retval = Hex(input)
  Ref = dec2hex(int(input(1)),2);
  M = dec2hex(int(input(2)),2);
  P = dec2hex(int(input(4)),2);
  N1 = dec2hex(round(input(6)),4);


  % Marks change to 24 bit FRAC output
  Frac = (input(6) - round(input(6)))*2^24;

  % Marks code to select optimal value with LSB=1
  % either floor (round down) or ceiling (round up) will
  % produce a value with LSB=1, unless a non fraction
  FracA = floor(Frac);
  if (mod(FracA,2) == 0)
    FracA = ceil(Frac);
    % if there was still no change after rounding then
    % force LSB to be true
    if (mod(FracA,2)==0)
      % this should be the worst case for error (1 bit out)
      FracA = FracA + 1;
    endif
  endif

  % change type before
  Frac = int(FracA);

  if Frac < 0
    Frac = 2^24 + Frac;
  endif

  N2 = dec2hex(Frac,6);

 Out = input(9);
  retval = {Ref,M,P,N1,N2,Out};
endfunction

retval = Hex(Frequencies(MPcomb(freq,RefVal(freq),Divider(freq)))); % RefVal now has an
argument
endfunction
```

CML Microcircuits

| | | |
|---|---|---|
| United Kingdom | p: +44 (0) 1621 875500 | e: sales@cmlmicro.com<br>techsupport@cmlmicro.com |
| Singapore | p: +65 62888129 | e: sg.sales@cmlmicro.com<br>sg.techsupport@cmlmicro.com |
| United States | p: +1 336 744 5050 | e: us.sales@cmlmicro.com<br>us.techsupport@cmlmicro.com |

www.cmlmicro.com