AM1-S100 Frequency Synthesizer

Part I

James A Crawford

Synopsis

The S100 frequency synthesizer is the first frequency synthesizer in a planned progression of synthesis projects on the drawing board. Although this synthesizer only provides core frequency coverage from 10 MHz through 1344 MHz, it also provides a wide compliment of precision reference outputs at 10 MHz and 100 MHz which can be used across my entire electronics laboratory as a master coherent source as needed.

The embedded 10 MHz precision frequency reference is based upon a 10 MHz OCXO which is frequency-locked to the 1 PPS output of a GPS receiver. The 100 MHz precision reference is created using a separate high quality 100 MHz VCXO which is phase-locked to the 10 MHz OCXO.

The tunable frequency source output is further conditioned using a bank of harmonic rejection filters and digital ALC for precise output level control.

1 Specifications

A high-level block diagram of the AM1-S100 reference portion is shown in Figure 1. The output harmonic filtering and amplitude leveling / control portion is shown separately in Figure 2. In terms of overall circuitry, it is rather striking how little the core frequency synthesizer device (LMX2571) takes up of the total real estate involved. The filter bank and leveling circuitry shown in Figure 2 will likely be used again in subsequent projects, however, so this time investment is still deemed worthwhile. Target specifications for the overall synthesizer are collected in Table 1.

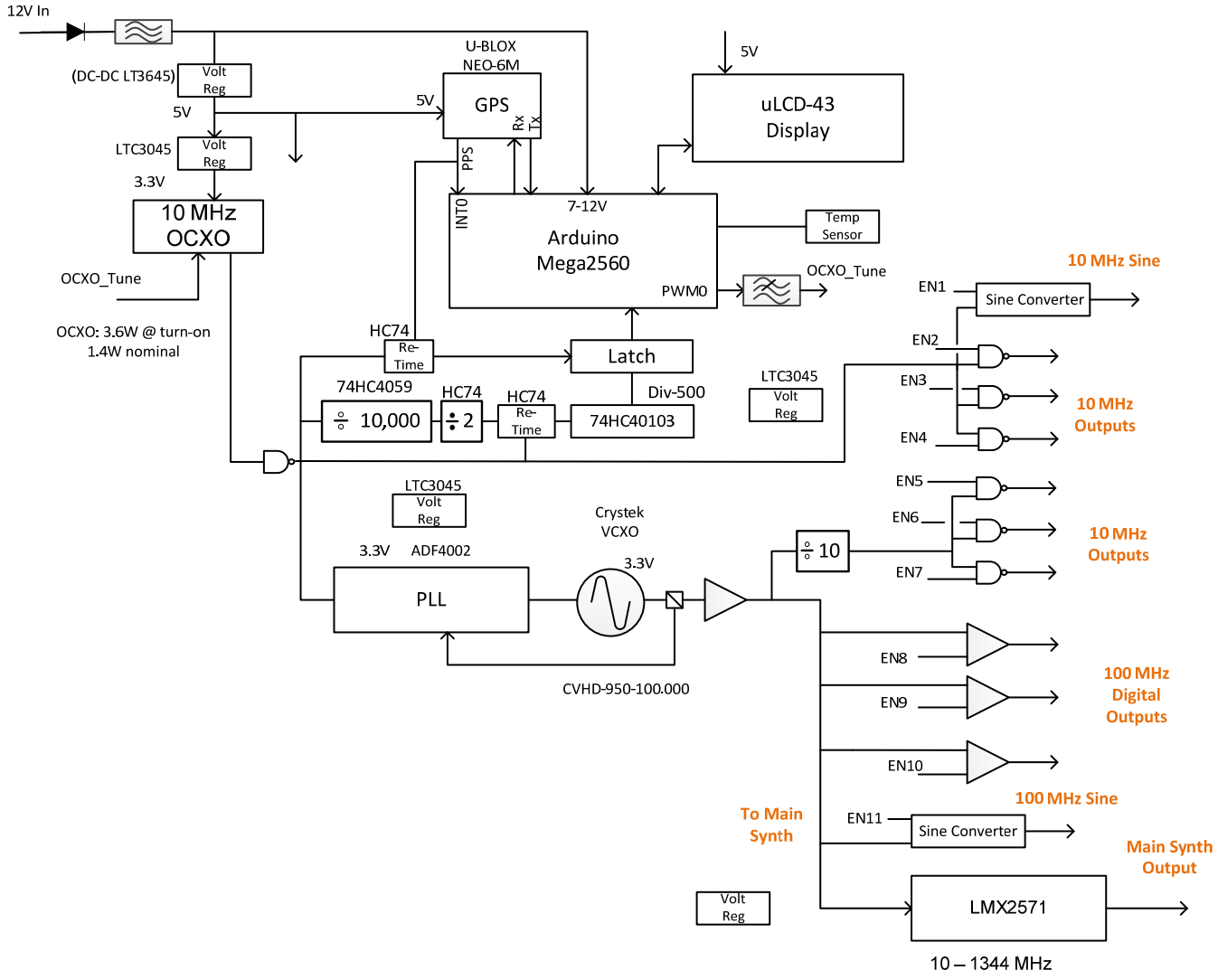

Figure 1 Frequency synthesis portion[1] of the AM1-S100

[1] From U23838 AM1-S100 Synthesizer.vsd.

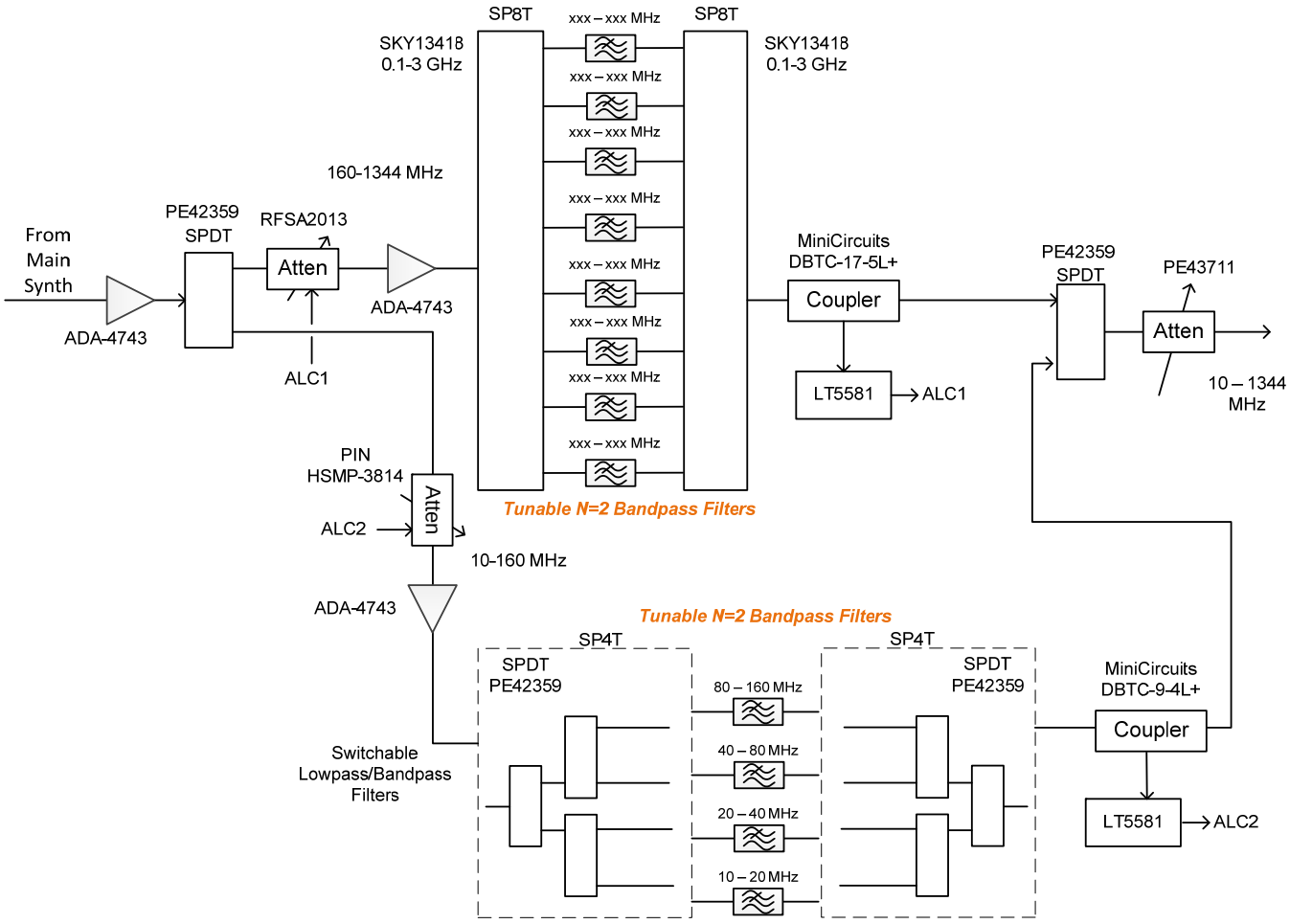

Figure 2 Output harmonic filtering and signal leveling portion[2] of the AM1-S100

In total, the AM1-S100 provides seven precision 10 MHz reference outputs using different signal families along with four precision 100 MHz reference outputs. This should be sufficient for any of the projects I foresee at the present time.

The remainder of this document looks at the anticipated performance for the S100 along with some outstanding conceptual issues that require separate attention.

[2] From U23838 AM1-S100 Synthesizer.vsd.

Table 1 High Level AM1 S100 Design Specifications

Specification	***Requirement***	***Comments***
Frequency Span	10 MHz – 1344 MHz	
Frequency Step Size f_{pd} = 100 MHz f_{pd}= 50 MHz	 ≥ 1.49 Hz ≥ 0.75 Hz	See §2 1075 MHz ≤ f_{out} ≤ 1344 MHz 1075 MHz ≤ f_{out} ≤ 1344 MHz
Output Power Range	+5 dBm to –25 dBm	30 dB range in 0.50 dB steps
Output Spurious Levels	< –75 dBc	
Output Harmonic Levels	< –50 dBc	
Frequency Accuracy Undisciplined GPS Disciplined	 ±0.5 ppm ±10 ppb	After appropriate warm-up period
Switching Time	≤ 100 μs	
Phase Noise Performance		Dictated almost entirely by the LMX2571's performance since a very clean 100 MHz reference will be provided to it.
Reference Outputs 10 MHz Sine Wave 10 MHz CMOS 10 MHz CMOS 100 MHz Sine Wave 100 MHz LVDS	 1 3 3 1 3	 Directly from OCXO “ Divided down from 100 MHz VCXO “ “
Input Supply Voltage(s)	10 ± 2V	Input voltage at circuit board level
Control		Keypad + LCD
Metrics		GPS locked GPS position Internal temperature Internal voltages OCXO tune voltage VCXO tune voltage ALC1 ALC2

2 Frequency Resolution

Frequency step size is one performance item requiring further attention. Even though the LMX2571 is a very capable frequency synthesizing device, it does have its limitations. Since substantial effort is being made in the overall S100 to deliver exceptionally good frequency precision (i.e., atomic standard level), it would be a shame to have the frequency resolution limited by a poor algorithm scheme. This section of the paper is focused entirely upon delivering the finest frequency resolution and finest frequency precision possible using the LMX2571.

Good phase noise performance is a priority item for the S100. To that end, the highest quality reference frequency possible is ideally used; in this case 100 MHz. Although the *R*-divider within the LMX2571 could be used to create lower reference frequencies thereby making some frequency synthesis cases achievable *exactly* using rational fractions, *R*-values greater than 5 will not be entertained so as to preserve phase noise performance.

An in-loop prescaler immediately follows the internal VCO as shown in Figure 3. Although the prescaler value can be programmed between 2 and 4, best phase noise performance is generally achieved using the divide-by-2 option and is consequently always used.

Another important limitation pertaining to frequency resolution is the fractional feedback divider which is limited to a fractional portion of 24 bits. This limits the fractional values to Num/Den where *Num* and *Den* are both integers that must be less than 2^{24}.

An additional important factor which directly enters into the frequency resolution discussion is the post-VCO output divider arrangement shown in Figure 4.

Putting these factors together results in the output frequency resolution being given by

$$\Delta f = \left(2 \frac{1}{R} \frac{Num}{Den} \frac{1}{Q} \right) f_{ref} \tag{1}$$

where the factor of 2 is due to the post-VCO in-loop prescaler, and *Q* is the post-VCO output divide ratio. Since all of the numerical quantities must be integer values, the achievable frequency step sizes must be a rational fraction of the fundamental reference frequency f_{ref}.

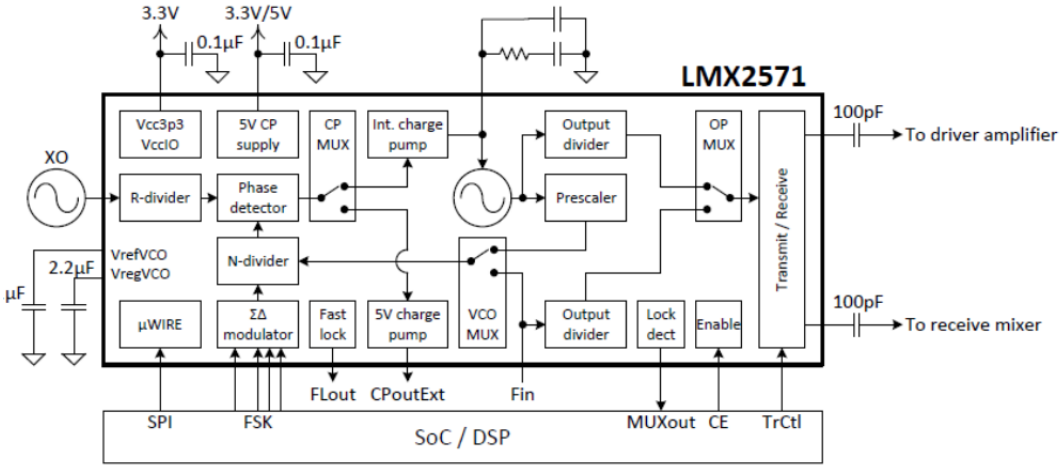

Figure 3 High level block diagram of the LMX2571 by Texas Instruments. Although the post-VCO prescaler is programmable from 2 to 4, phase noise performance is generally better using a value of 2 and is consequently always used.

Figure 4 Post-VCO output divide chain within the LMX2571

The VCO core operating frequency range spans from 4300 MHz through 5376 MHz. Since the minimum CHDIV1 value in Figure 4 is 4, this means that the highest output frequency range covers from 1075 MHz through 1344 MHz before CHDIV1 must be increased to 5. This may be a significant fact in some designs since the output harmonic performance for odd-valued CHDIV1 values will be noticeably worse than for even-values.

2.1 Obtaining Best Frequency Resolution from the LMX2571

The main frequency reference provided to the LMX2571 is 100 MHz. Even though the fractional-N denominator used within the LMX2571 can be as large as $2^{24} - 1$, tuning a specific frequency with *absolute accuracy* is not always as easy as one would like while simultaneously using the largest phase-comparison frequency possible for best phase noise performance. An example will bear this issue out more clearly.

Assume that a frequency offset of 1,342,111 Hz is desired relative to an integer multiple of 100 MHz and that 100 MHz is being used as the phase comparison frequency. For the moment, assume that the post-VCO output divide ratio is 4 so that the frequency offset at the VCO output is 5,368,444 Hz. The associated fraction is

$$frac = \frac{5,368,444}{1e8} \approx 0.05368444 \tag{2}$$

Since the fraction times 2^{24} is not an integer value, it is impossible to synthesize this frequency offset (5,368,444 Hz) exactly using the 100 MHz reference frequency and a denominator value of $2^{24} - 1$. The best approximate answer to this question can be found by expanding the fractional value in (2) as a continued fraction as

$$0.05368444 \approx \cfrac{1}{18+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{6+\cfrac{1}{4+\cfrac{1}{1+\cfrac{1}{2+\ldots}}}}}}}}} \tag{3}$$

With the (truncated) continued fraction terms shown in (3), a very close approximation in rational form can be assembled as

$$0.05368444 \approx \frac{1,739}{32,393} \tag{4}$$

which exhibits a residual error of about 2e-9 which amounts to about 0.2 Hz at the VCO output. Since the denominator in (4) is still much less than 2^{24}, an even better approximation is likely possible by extending the continued fraction even further. Doing so does of course require high-precision divide operations.

Striving for such precision may seem like an academic exercise, but with the care taken to use a GPS-disciplined OCXO and subsequent VCXO in the design, this effort is warranted. The calculation done in (3) is straight-forward to apply: invert the fraction, retain the integer portion as a leading factor, take the new fractional portion and repeat the operations again. If the (new) fractional portion ever equals zero, halt the process recognizing that the associated rational result is an exact answer. Working backwards to the fraction in (4) may seem to be rather laborious but the following recursion makes this process fairly straight forward. As an example, consider the simple continued fraction given by

$$frac = \cfrac{1}{a + \cfrac{1}{b + \cfrac{1}{c}}} \tag{5}$$

Note that $b + \frac{1}{c}$ can be rewritten as $\frac{bc+1}{c}$ which is easily inverted to convert (5) into

$$frac = \cfrac{1}{a + \cfrac{d}{e}} \tag{6}$$

where $d = 1 + bc$ and $e = c$. The same steps can be performed repeatedly on the denominator portion of (6). In the context of a larger continued fraction like

$$frac = \cfrac{1}{D_1 + \cfrac{1}{D_2 + \cfrac{1}{D_3 + \cfrac{1}{D_4 + \cfrac{1}{D_5 + \cfrac{1}{D_6 + \cfrac{1}{D_7 + \cfrac{1}{D_8}}}}}}}} \tag{7}$$

the associated recursion starts with

$$\begin{aligned} D_{new} &= D_7 \\ N_{old} &= 1 \\ D_{old} &= D_8 \\ k &= 7 \end{aligned} \tag{8}$$

followed by

$$\begin{aligned} N'_{old} &= D_{old} \\ D'_{old} &= N_{old} + D_{old} D_{new} \end{aligned} \tag{9}$$

and then

$$\begin{aligned} N_{old} &= N'_{old} \\ D_{old} &= D'_{old} \\ k &= k-1 \\ D_{new} &= D_k \end{aligned} \tag{10}$$

Equations (9) and (10) are repeated over and over until k = 0 at which point the sought after fraction is given by N_{old} / D_{old}.

For the example at hand (4), it turns out that an exact result is possible while still using a denominator value less than 2^{24} as

$$0.05368444 = \frac{145,539}{2,711,009} \tag{11}$$

The Excel worksheet associated with (3) is shown in Table 2. The Excel worksheet associated with (10) and (11) is shown in Table 3.

Table 2 Excel Worksheet Associated With (3)

0.05368444000000	
18.62737136	18
1.593952271	1
1.683636967	1
1.462764666	1
2.16092557	2
6.214052865	6
4.671743119	4
1.488664299	1
2.046394634	2
21.55421677	21
1.804348152	1
1.243242739	1
4.111119626	4

Table 3 Excel Worksheet Associated With (10) and (11)

Dnew	Nold	Dold
1	1	4
1	4	5
21	5	9
2	9	194
1	194	397
4	397	591
6	591	2761
2	2761	17157
1	17157	37075
1	37075	54232
1	54232	91307
18	91307	145539
Answer	145,539	2,711,009

The AM1-S100 uses an Arduino MEGA2560 as its main processor. The Arduino MEGA is an 8-bit computational engine and performing high-precision arithmetic is not its real forte. Fortunately, a *big number* library has been previously assembled[3] for the Arduino MEGA2560 and this hurdle has already been tackled. An Arduino MEGA2560 script to calculate *e* to about 100 decimal places is shown below for instance.

```
#include "BigNumber.h"
static int DIGITS = 103;

void print_bignum (bc_num x)
{
  char *s=bc_num2str(x);
  Serial.println (s);
  free(s);
}

void setup ()
{
  Serial.begin (115200);
  Serial.println ();
  Serial.println ();
  bc_init_numbers ();  // initialize library

  char buf [10];

  // some big numbers
  bc_num n= NULL, e = NULL, one = NULL;
```

[3] BigNumber.cpp by Nick Gammon, 22nd January 2013. Downloadable at http://www.gammon.com.au/Arduino/BigNumber.zip .


```
  // the number: 1
  bc_str2num (&one, "1", DIGITS);

  e = bc_copy_num (one);
  n = bc_copy_num (one);

  for (int j = 1; j <= 200; j++)
  {
    bc_num E = bc_copy_num (e);
    bc_num i = NULL;
    sprintf (buf, "%i", j);
    bc_str2num(&i, buf, DIGITS);
    bc_multiply (i, n, &n, DIGITS);  // n is i! (factorial of i)
    bc_free_num (&i);
    bc_num inverse = NULL;
    bc_divide (one, n, &inverse, DIGITS);
    bc_add (inverse, e, &e, DIGITS);
    bc_free_num (&inverse);
    if (bc_compare (e, E) == 0)  // sequence has converged
      break;
    bc_free_num (&E);
  }

  print_bignum (e);
} // end of setup

void loop () { }
```

Result:

```
2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642742 40
```

Referring back to (1), the output divide ratio *Q* is dictated by the output frequency and is consequently not a free parameter. The output frequency ranges and corresponding value of *Q* are partially summarized in Table 4.

Table 4 Frequency Limits Versus Output Divide Ratio Q

Q	**f_{vco}, MHz, min**	**f_{vco}, MHz, max**	**f_{min}, MHz**	**f_{max}, MHz**
4	4300	5376	1075	1344
5	4300	5376	860	1075.2
6	“	“	716.666	896
7	“	“	614.2857	768
8	“	“	537.5	672
10	“	“	430	537.6
12	“	“	358.3333	448
14	“	“	307.1429	384
16	“	“	268.75	336
…				
448	“	“	9.5982	12

An important question in the context of Table 4 is the degree of frequency coverage when using only even output divide ratios. As shown in Table 5, only two of the many frequency ranges use odd-valued output divide ratios, but they are nevertheless two important ranges. Since the best spurious and best phase noise performance is usually obtained using the maximum output divide ratio *Q*, it is advantageous to use the output frequency versus Q-value arrangement as shown in the far-right columns of Table 5.

Table 5 Only 2 of the Possible Output Divide Ratios[4] are Odd (But They Are Important Ones)

	Maximum Coverage			Contiguous Coverage Using Maximum Q Value					
Q	**fmin, MHz**	**fmax, MHz**		**fvco,min**	**fvco,max**		**fout,min**	**fout,max**	
4	1075	1344		4300.8	5376		1075.2	1344	
5	860	1075.2		4480	5376		896	1075.2	
6	716.6666667	896		4608	5376		768	896	
7	614.2857143	768		4704	5376		672	768	
8	537.5	672		4300.8	5376		537.6	672	
10	430	537.6		4480	5376		448	537.6	
12	358.3333333	448		4608	5376		384	448	
14	307.1428571	384		4704	5376		336	384	
16	268.75	336		4300.8	5376		268.8	336	
20	215	268.8		4480	5376		224	268.8	
24	179.1666667	224		4608	5376		192	224	
28	153.5714286	192		4704	5376		168	192	
32	134.375	168		4300.8	5376		134.4	168	
40	107.5	134.4		4480	5376		112	134.4	
48	89.58333333	112		4608	5376		96	112	
56	76.78571429	96		4704	5376		84	96	
64	67.1875	84		4300.8	5376		67.2	84	
80	53.75	67.2		4480	5376		56	67.2	
96	44.79166667	56		4608	5376		48	56	
112	38.39285714	48		4704	5376		42	48	
128	33.59375	42		4300.8	5376		33.6	42	
160	26.875	33.6		4480	5376		28	33.6	
192	22.39583333	28		4608	5376		24	28	
224	19.19642857	24		4704	5376		21	24	
256	16.796875	21		4300.8	5376		16.8	21	
320	13.4375	16.8		4480	5376		14	16.8	
384	11.19791667	14		4608	5376		12	14	
448	9.598214286	12		4300	5376		9.598214	12	

Exact frequency tuning is an important capability for this project. Exact frequency steps sizes of 1 Hz and 5 Hz (or any integer multiple thereof) is possible using the 100 MHz reference frequency except for one output divide ratio *Q* as shown in Table 6. This statement is true if the maximum reference divide ratio is limited to 5. If the maximum reference divide ratio is increased to 8 (implying a phase-comparison frequency of 12.5 MHz), however, no gaps occur in Table 6.

For potentially odd-ball frequencies which may be required (sub-Hz resolution), the approximate continued-fraction method (7) is the only recourse available.

[4] From U24881_LMX2571_Frequency_Coverage.xlsx.

Table 6 Divide-by-R Value and Minimum Fractional Denominator[5] for Exact ∆f Step-Size. (The Div-R value must be increased to 8 to have 1 Hz coverage for the *Q* = 7 case.)

	df= 1 Hz		df= 5 Hz	
Q	Div-R	Den (min)	Div-R	Den(min)
4	2	12,500,000	1	5,000,000
5	2	10,000,000	1	4,000,000
6	4	12,500,000	1	10,000,000
7	1	***NONE***	2	10,000,000
8	1	12,500,000	1	2,500,000
10	1	10,000,000	1	2,000,000
12	2	12,500,000	1	5,000,000
14	4	12,500,000	1	10,000,000
16	1	6,250,000	1	1,250,000
20	1	5,000,000	1	1,000,000
24	1	12,500,000	1	2,500,000
28	2	12,500,000	1	5,000,000
32	1	3,125,000	1	625,000
40	1	2,500,000	1	500,000
48	1	6,250,000	1	1,250,000
56	1	12,500,000	1	2,500,000
64	1	1,562,500	1	3,125,000
80	1	1,250,000	1	250,000
96	1	3,125,000	1	625,000
112	1	6,250,000	1	1,250,000
128	1	781,250	1	1,562,500
160	1	625,000	1	125,000
192	1	1,562,500	1	312,500
224	1	3,125,000	1	625,000
256	1	390,625	1	78,125
320	1	312,500	1	62,500
384	1	781,250	1	15,625
448	1	1,562,500	1	312,500

2.2 GPS Receiver for 1 PPS and Outstanding Frequency Accuracy

It has become fairly common practice to provide a precision one pulse per second (PPS) output even from fairly inexpensive GPS products. For example, I just purchased[6] a Ublox NEO-6M GPS receiver from Amazon for about $20 which has a dedicated 1 PPS output[7] with seemingly excellent performance.

A frequency-locked approach is used in the S100 in order to impress GPS frequency accuracy upon the OCXO in Figure 1. The selected OCXO is an AOCJY 10.00 MHz from Abracon[8] which exhibits excellent long term stability and aging characteristics. The initial frequency accuracy is not, however, specified which likely puts it in the range of ±0.5 ppm which translates to ±5 Hz at 10 MHz and 672 Hz at the 1344 MHz maximum synthesizer output setting.

[5] Computed in u24882_convenient_freq_res.m and assembled in u24881_LMX2571_Frequency_Coverage.xlsx.

[6] https://www.amazon.com/gp/product/B01AW5QYES/ref=oh_aui_detailpage_o02_s00?ie=UTF8&psc=1

[7] The Ublox LEA-6T modules can be configured for a wide range of output pulse frequencies using the UBX-CFG-TP5 message.

[8] Purchases from Digikey for about $120.

The 1 PPS output limits the unambiguous frequency range to at most ±0.5 Hz of course due to the Nyquist criterion. In order to slave the OCXO's output frequency to that of the GPS satellite system, the OCXO's output frequency must be appropriately handled.

Since the S100 is intended to provide excellent frequency reference stability even if it is not being used as a tunable RF source, it is desirable to use a hardware configuration to frequency-lock the OCXO to the GPS receiver's 1 PPS output without always involving the Arduino MEGA2560. The baseline concept used in the S100 is shown in Figure 5.

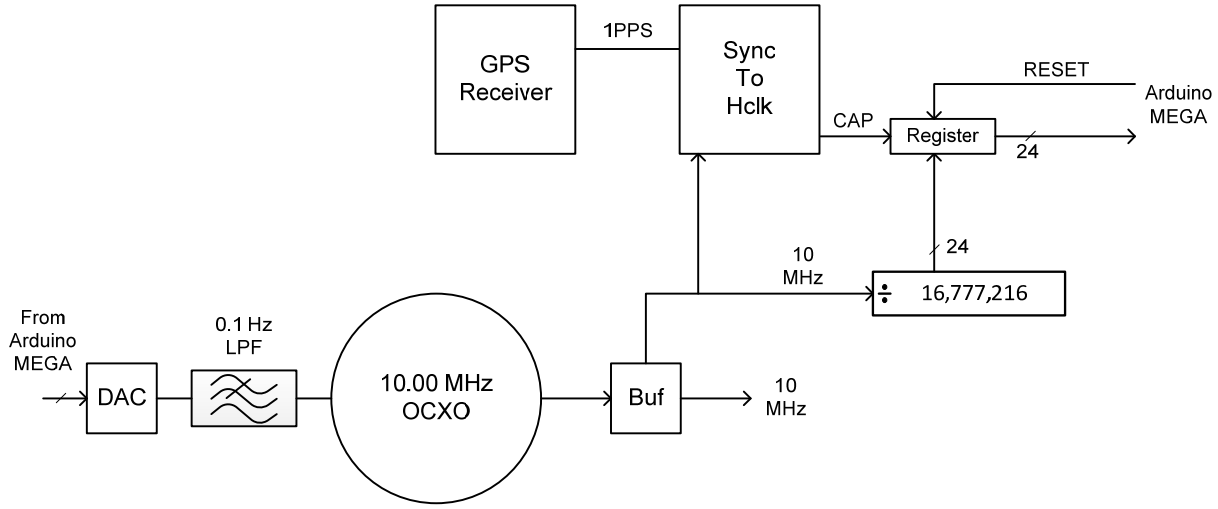

Figure 5 Frequency-locked concept used to slave the 10 MHz OCXO to GPS precision timing

The frequency discriminator function in this concept is performed by the Arduino MEGA since numerical computations are so much easier to perform with it rather than have dedicated hardware to do so.

The 1PPS signal and 10 MHz OCXO outputs shown in Figure 5 are generally asynchronous with respect to one another. The first operational step in making a frequency measurement is therefore to align the 1PPS signal with the 10 MHz high speed clock thereby side-stepping potential metastability issues. The result is the synchronized capture signal (CAP) shown in the figure.

If the 10 MHz OCXO is operating at precisely 10 MHz, the 24-bit counter should accumulate exactly 10,000,000 counts in 1 second. The 1PPS signal does have some time-jitter associated with it, however, on the order[9] of ±40 nsec which alone will cause the counter value to vary on the order of 0.4 ticks per 1 second interval even if everything else is *perfect*. Since the counter is free-running with the OCXO's output, working with the Arduino Mega, fractional ticks can be accumulated over 10 seconds, 100 seconds, or any time interval of interest for finer frequency resolution.

The frequency discriminator calculation performed by the Arduino MEGA amounts to computing the 24-bit register residual as

$$\Delta C_{k,m} = \text{modulo}\left(C_k - C_{k-m}, 2^{24}\right) = \sum_{p=m}^{k} \text{modulo}\left(C_p - C_{p-m}, 2^{24}\right) \tag{12}$$

where index k corresponds to the number of seconds since the measurement process was initiated, and C_k is the 24-bit counter value for the k^{th} capture. Multiple values of (12) can be computed by the Arduino MEGA corresponding to different baseline measurement times (in terms of seconds) based upon parameter m in order to accelerate the frequency pull-in process.

For example, assume that we ultimately want the OCXO's frequency precision to be slaved to GPS time to within 0.01 ppm. At 10 MHz, we would have to wait a minimum of 10 seconds in order to see

[9] §8.3.3 of U24800, "GPS Essentials of Satellite Navigation," Ublox.

a count-error of 1. Normally, it is desirable to operate on a greater error-value than one, however, so the measurement would ideally like to see a count-error more on the order of at least 10. If the objective was 0.0001 ppm though, the measurement time base would have to be increased to at least 1,000 seconds to see a count-error of just one which is a long time to wait, particularly if the OCXO's frequency error is substantial to start with.

The Arduino MEGA's frequency discriminator function therefore keeps track of ΔC values corresponding to *m* = {1, 10, 20, 40, 80, 160, 320, 640, 1280} and uses a *smart algorithm* to correct the OCXO's frequency in the most time-expedient / lowest measurement error variance manner possible.

Of course at some point with increasing *m*, other imprecisions enter into the picture and it serves no purpose to keep increasing *m*. The OCXO has its own aging and drift rate which also comes into play as *m* is increased, for instance. The Arduino MEGA software will be written with sufficient hooks to examine these details as time permits.

As the Arduino MEGA makes estimates for the OCXO frequency adjustments necessary, it applies them by writing a D/A word which adjusts the OCXO's output frequency as shown in Figure 5. The D/A is external to the Arduino MEGA and is configured to retain its programmed value even after power is removed from the Arduino. Adjusting the OCXO's tuning voltage does marginalize the previous ΔC values computed, but not entirely. If the OCXO's frequency tuning characteristic is perfectly linear for example, most of the ΔC information would still harbor value going forward. Pursuit of these details involves estimating the OCXO's tuning sensitivity as well as the residual frequency error in an optimal manner as discussed in a future project report.

3 Output Harmonic Filter Bank

The low frequency tunable filters shown in Figure 2 were the subject of several earlier papers[10]. The tunable filters covering center frequencies above about 400 MHz will be fabricated using a hybrid combination of lumped-element and discrete components in order to retain good stopband performance.

Automatic level control (ALC) shown in Figure 2 is used to deliver an accurate power level at the RF output. Internal signal levels are measured using a pair of LT5581 precision detectors and gain adjustments are made for the high frequency range using a RFSA2013 variable attenuator whereas the adjustments are made using a HSMP-3814 PIN diode based attenuator for the lower frequencies. Since power level variations will be very slow with respect to time, and some degree of power level versus frequency calibration may be required, the ALC loops will be closed through the Arduino MEGA2560 used for the project.

4 Packaging

The finished product needs to have reasonable attention given to packaging aesthetics and yet keep related costs acceptably low. Two views of the target packaging are shown in Figure 6 and Figure 7.

The top-front of the enclosure has a sturdy aluminum plate as shown which is good for structural integrity but lacks the polish of a more commercial product. One improvement would be to anodize the aluminum panel before engraving it with my CNC machine. Color and feature choices would, however, still be quite limited. The baseline plan at present is to overlay the aluminum plate with a thin Plexiglas piece which has been painted and then engraved on its back-side. Once these steps have been completed, the final perimeter will be cut using the CNC machine as well. This approach will keep the outside exposed surface of the panel completely smooth while offering design flexibility.

Electromagnetic interference (EMI) issues will be handled by using Faraday shields within the plastic enclosure and therefore out of sight.

[10] U24143 Versatile BPF with Tunability, Parts I, II, and III.

Figure 6 Target packaging front view

Figure 7 Target packaging back panel view

5 Next Steps

The next project steps are focused on doing the detailed design for Figure 1 and Figure 5, followed by circuit board design, fabrication, and check-out. Some Arduino MEGA software will also be involved, as well as other software to drive its associated liquid crystal display. This should all be great fun, and I am already off and running!