(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 July 2009 (16.07.2009)

PCT

(10) International Publication Number
WO 2009/089321 A2

(51) International Patent Classification: Not classified

(21) International Application Number:
PCT/US2009/030398

(22) International Filing Date: 8 January 2009 (08.01.2009)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/019,811 8 January 2008 (08.01.2008) US

(71) Applicant *(for all designated States except US)*: **ESS TECHNOLOGY, INC.** [US/US]; 48401 Fremont Boulevard, Fremont, California 94538 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **MALLINSON, Andrew, Martin** [CA/CA]; 1306 Huckleberry Road, Kelowna, B.C. VIP 1M5 (CA).

(74) Agent: **SUZUE, Kenta**; Haynes Beffel & Wolfeld LLP, P.O. Box 366, Half Moon Bay, California 94019 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

(54) Title: DIGITAL FEEQUENCY GENERATOR

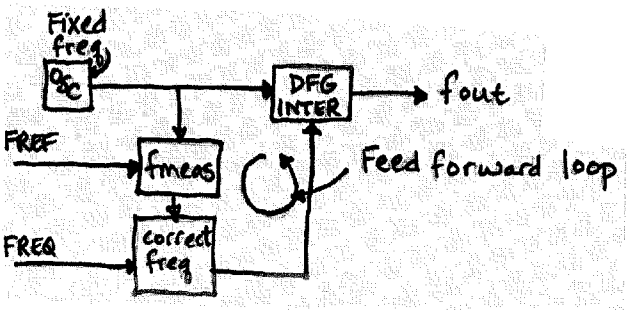

Figure 2

(57) Abstract: Discussed herein are a feed-forward control loop, 'almost-binary' counter, and ring oscillator.

WO 2009/089321 A2

DIGITAL FREQUENCY GENERATOR

SUMMARY

[0001] Various embodiments of the technology are disclosed, such as a feed-forward control loop circuit, an "almost-binary" counter, a longer-than-necessary ring oscillator.

[0002] The feed-forward control loop circuit generates a fixed and predictable output frequency despite a possibly variable and unpredictable source.

[0003] The "almost-binary" counter can calculate a number of states between two states of a counter such as a Pseudo-Random Binary Sequence Counter (PRBS) or a Linear Feedback Shift Register (LFSR).

[0004] The ring oscillator to resolve sampling timing error.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] Figure 1 is a block diagram of a feedback digital frequency generator.

[0006] Figure 2 is a block diagram of a feed-forward digital frequency generator.

[0007] Figure 3 is a block diagram of a feed-forward digital frequency generator with a filter to increase resolution.

[0008] Figure 4 is a block diagram of a multiple channel feed-forward digital frequency generator with a filter to increase resolution.

[0009] Figure 5 is a block diagram of a ring oscillator with multiple outputs representing multiple phases of the oscillator.

[0010] Figure 6 is a trace of a periodic ramp, with each period representing the phase of one oscillator period, and multiple external events occurring at different phases of different periods.

[0011] Figure 7 shows the trace of Figure 6, aligned with a trace of a clock, illustrating the time delay between the oscillator and a clock derived from the oscillator.

[0012] Figure 8 shows the traces of Figure 7, aligned with a first trace of a first sample of an external event S0 unsynchronized with the oscillator derived clock, and a second trace of the synchronized event S0' of event S0 registered into the clock domain and sampling the oscillator again.

[0013] Figure 9 resembles the traces of Figure 8, but counts the phases with a three-state counter, distinguishing the phases as phase 1, phase 2 or phase 3, thereby removing the ambiguity in phase difference between the unsynchronized and synchronized events.

[0014] Figure 10 is a block diagram of a ripple carry counter.

[0015] Figure 11 is a block diagram of a ripple carry counter clocking the elements on the same clock edge.

[0016] Figure 12 is a block diagram of a ripple carry counter clocking the elements on the same clock edge, which stops the carry in every bit cell.

[0017] Figure 13 is a table listing the count sequence of the "S"-state DTypes in Figure 12.

[0018] Figure 14 is a block diagram of a ripple counter with "C"-state DTypes which indicate whether the following "S"-state DType should toggle on the next clock edge, representing an improvement upon the block diagram in Figure 12 due to the re-positioning of the AND gate such that only one gate delay (of AND or Exclusive-Or) is interposed between each DType.

[0019] Figure 15 is a table listing the count sequence of the "C"-state and "S"-state DTypes in Figure 14.

DETAILED DESCRIPTION

[0020] Figure 1 is a block diagram of a feedback digital frequency generator ("DFG"). This circuit includes feedback, dividers N and M, and the element marked as "DFG INTER" which allows the use of a fixed frequency oscillator. Some advantages of this configuration are that the loop is first-order and hence unconditionally stable, and that the phase detector is entirely digital and so can run arbitrarily slowly without degradation. (This latter aspect means that N and M can be essentially unlimited in value allowing flexible and accurate frequency setting. The significance of allowing N and/or M to be a large number is that this is directly related to the accuracy of the frequency setting: consider that N may be 1,000,000 and M may be 1,000,000 and adjustable to 1,000,001 – this is changing the output frequency by one part in a million, or 1 ppm. If M were smaller, say 100, the minimum adjustable change of output frequency would be only 1% - far less). However, a potential problem of this DFG loop is that, if the nominally fixed frequency oscillator drifts (due to its inherent phase noise, or due to temperature or voltage variation), that drifting oscillator cannot be corrected at a rate any faster than the rate of output signals from the box labeled "Fcomp". In other words, although the Fcomp element can run arbitrarily slowly and N and/or M can therefore be large, this DFG exhibits a problem similar to that of a conventional PLL: there is degradation in the phase noise as the operating frequency of Fcomp is reduced.

[0021] Figure 2 is a block diagram of a feed-forward digital frequency generator. This degradation of phase noise may be mitigated by this feed-forward configuration. The rate of the fixed frequency oscillator is continually measured by the block labeled "Fmeas" which makes

use of the reference frequency so that it can calculate the precise frequency of the fixed frequency oscillator. A second block labeled "correct freq" then takes the output from the "Fmeas" block and uses it to set the control value to the "DFG INTER" block. The output of the DFG INTER block is not used in the control circuit; the output of that block need not be inspected since both its input frequency is known (Fmeas has measured it) and its control value is known ("correct freq" has created it). Any variation in the nominally fixed frequency of the oscillator is captured by the "Fmeas" block and immediately used to adjust the control value to the DFG. The rate of correcting any frequency drift in the oscillator is decoupled from the required frequency – the oscillator drift can be corrected at a rate of, for example, 10MHz by the Fmeas block without impact on the accuracy of the output frequency setting. Furthermore, this is a feed forward configuration and consequently has no stability issues and instantly responds to changes to the frequency request input.

[0022] Figure 3 is a block diagram of a feed-forward digital frequency generator with a filter to increase resolution. Depending upon the rate and resolution that the Fmeas block can achieve, it may be advantageous to add a filter to the output of the Fmeas block. This filter reduces the rate at which the fixed frequency oscillator phase drift can be corrected, but has the advantage that it increases the resolution (and the hence the frequency setting accuracy) of the configuration.

[0023] Figure 4 is a block diagram of a multiple channel feed-forward digital frequency generator with a filter to increase resolution. Shown here is the overhead per channel of one "DFG INTER" block and one "correct freq" block. The configuration requires only one instance of the "Fmeas" block (and one instance of the filter on its output if desired) to support any number of channels, though more instances of the "Fmeas" block and the filter may be optionally added. In one embodiment an optional filter that may be enabled e.g. under software control in the path between the measurement unit ("Fmeas") and the calculation block ("correct freq").

[0024] Consider the following example illustrating the role of the "Fmeas" block in the feed-forward implementation of the DFG. Assuming the free-running oscillator is about 1Ghz, and assuming it is desired to correct the frequency at a rate of 1Mhz, then the "Fmeas" block must complete its task in 1uS. But in a 1uS interval only 1000 cycles of the oscillator have passed, so how then can the "Fmeas" block derive an answer that is any more accurate than 1 part in 1000? Therefore, there is a compromise to be made: the faster we ask the Fmeas unit to operate, the less accurate it becomes.

[0025] This compromise can be broken. In one embodiment, we measure the frequency of the free running oscillator to within a fraction (in fact 1/10th) of the oscillator cycle. So, in this example the "Fmeas" configuration would resolve to 1 part in 10,000; ten times better than we may expect.

[0026] The following discusses the oscillator and frequency measurement unit. In order to subdivide the oscillator periods the oscillator is built as a "ring" of an odd number of inverting elements, the output of one element being connected to the input of the next to make an unstable circuit that oscillates (since there are an odd number of elements the collection cannot settle to a fixed pattern – the output after propagating round the ring always comes back not matching the input). If, when the FREF edge arrives from the reference input frequency, we could somehow sample the oscillator and say, for example, on that edge the oscillator was at 3.7 cycles; then on the next edge of the FREF input sample again and find that the oscillator was at 55.3 cycles, we would know that 51.6 cycles had passed. Clearly, being able to resolve a fraction of a cycle has increased the accuracy with which we can measure the oscillator frequency in any given elapsed time. Figure 5 is a block diagram of a ring oscillator with multiple outputs representing multiple phases of the oscillator. The oscillator is shown as five inverting elements, the output is taken from one of the five, but DTypes (or D flip flops) are provided that can sample the state of the ring when an external event occurs. Another common example uses 3 elements.

[0027] Figure 6 is a trace of a periodic ramp, with each period representing the phase of one oscillator period, and multiple external events occurring at different phases of different periods. Using this method we see that in principle we can find where in the cycle of the oscillator an event has occurred. Figure 6 represents the phase of the oscillator and the samples that may be taken for two external events S0 and S1. We can imagine that the S1 event for example, has occurred at about 70% of the way through the cycle represented at the center of the drawing. There is however, a practical problem – the internal clock derived from this oscillator has a finite delay – it does not occur precisely on the phase boundaries of the oscillator.

[0028] Figure 7 shows the trace of Figure 6, aligned with a trace of a clock, illustrating the time delay between the oscillator and a clock derived from the oscillator. The signal S1 which in prior diagrams appeared to be at about 70% through the cycles is seen to be at about 40% through the cycle due to the delay from the oscillator to the internal logic clock signal. This delay from the oscillator to the internal clock is inevitable because the clock is running so relatively fast: we can imagine that one cycle is about 1nS, so a delay of as little as 100pS is a 10% error in the position within the phase. How can we make the circuit such that the event is

measured as a fraction of the cycle of the internal clock – not as a fraction of the cycle of the oscillator (since these are slightly offset in time)?

[0029] We sample the oscillator state a second time – this second time with a signal that is derived relative to the clock, not relative to the external event. To do this we simply register the external event into the clock domain, or have the external event on the D input of a Dtype clocked by the clock. After the external event occurs, the Q output of the DType will show the state. This Q output changes relative to the clock. Figure 8 shows the traces of Figure 7, aligned with a first trace of a first sample of an external event S0 unsynchronized with the oscillator derived clock, and a second trace of the synchronized event S0' of event S0 registered into the clock domain and sampling the oscillator again.

[0030] The external event is shown causing a sample of the oscillator S0. That same event is then clocked into the clock domain by a DType and the output is used to sample the oscillator again at S0'. In principle the samples taken in the clock domain should be always the same number (since the output Q is always a certain time after the clock and hence always at the same phase of the oscillator). That second sample therefore serves as a reference to the phase of the oscillator as seen by the clock. The actual phase of the input signal is the number S0'-S0 (ie the difference of the phase). However many cycles of the clock have passed say N+1, but in the earlier cycle by S0'-S0 – we can find therefore the time as N plus some fraction. Although theoretically possible this proposal also has a problem: the calculation of S0'-S0 cannot be done in the rare case where the S0 event occurs in the prior cycle very close to the S0' event. The precise reason for this is due to uncertainty (jitter) in the timing which is best explained by an example: let us assume that the S0' event has been taken and has yielded a phase of 60% (this is close to the example in the drawing above), and the S0 event has registered 59%. Now, does that mean that the S0 event occurred very shortly before the S0' event as we would suppose? It may not be known – theoretically, if all were perfect it has to be the 59% in the same cycle as S0', because the DType just captured it. But in practice no DType can respond instantly and it is always possible to find a time of indecision where we cannot know if the S0 was 59% in the prior cycle or 59% in the same cycle as the 60% S0'. Consequently, another change is needed to solve this problem. That change is to make the oscillator longer than the equivalent oscillator such that the phase differs in the adjacent cycles.

[0031] Figure 9 resembles the traces of Figure 8, but counts the phases with a three-state counter, distinguishing the phases as phase 1, phase 2 or phase 3, thereby removing the ambiguity in phase difference between the unsynchronized and synchronized events. To achieve this differentiation the oscillator is made up of 15 sections with logic attached such that three

output cycles are generated during one cycle of the 15 sections. Use of this three times-longer oscillator ensures that even in the presence of a staticizer failure (meta-stability) in the sampling DType there is no error generated when processing the two sample points. It remains now only to count the actual cycles of the compound oscillator and add that cycle difference to the fraction calculated from these two samples.

[0032] The following discusses a high speed "almost binary" counter. A DFG embodiment makes use of a Pseudo-Random Binary Sequence Counter (PRBS. Often called perhaps more descriptively a Linear Feedback Shift Register or LFSR) to achieve a very high count rate. A LFSR has very small and simple "next-state generation" logic – a single Exclusive OR between two of the sections – but it suffers from an essentially random sequence of states that are quite difficult to process in a conventional arithmetic logic unit (ALU), so that it is not possible to conveniently calculate how many cycles lie between any two states of a LFSR. This "Almost Binary" counter has the same high speed properties as the LFSR due to the very simple next state logic, but unlike the LFSR the sequential states are relatively easy to process with a conventional ALU.

[0033] Suppose you want to count edges of a high speed clock – let's imagine that the clock is 1.5GHz and the logic you are using is 0.15u CMOS. A Single Dtype can toggle at 1.5Ghz but making a counter of say 10 bits length run at this speed is not easy (it may not even be possible) because the delay accumulated in the carry path (from the LSB to the MSB at the transition from 0111111111 to 1000000000) is longer than the clock cycle. How then can such a counter be made? If you give the problem to the synthesis tool – it has been programmed with many "tricks" to solve this problem – the speed limitation will prevent the elaborator from using a ripple through or propagate/generate carry and so it will switch to a parallel generation of the carry to the MSB block. In other words, it will analyze the state in a multi-input combinatorial block that can predict that a carry to the MSB will occur. This is a fast means to make a conventional binary counter but at some point even this method fails because the depth of the parallel carry generator logic increases to the point where it cannot deliver the carry bit in time. For a conventional binary count sequence, a ROM can be used to look up the carry or indeed to look up the entire next state, the limitation then being the ROM access time. Non-binary count sequences can be very fast: the LFSR "Linear Feedback Shift Register" (or PRBS "Pseudo Random Binary Sequence") techniques require only adjacent bits to generate the next state of a local bit, but this method suffers from a problem that the count sequence cannot be operated upon by processing logic (you cannot simply subtract two counts to see the difference for

example) since the sequence is essentially random. LFSR techniques are in common use for very high speed "counting" of events.

[0034] This note outlines a counter that executes a count sequence that is "close to binary" and provides a simple means to convert the state of the counter to a binary count, alleviating the problem of increasing carry propagation delay. The counter here described can count at very high rates – to within a fraction of the Dtype toggle rate, and operations such as the difference of counts can be done with conventional signal processing.

[0035] One method for building a binary counter is easy to understand, it corresponds to the way the odometer in the car works: whenever the lower significance digit makes a transition back to the zero state, increment the next higher significance digit. So for example, from the state 19 to 20 the rule says that the 9 just went to 0 so now increment the 1 from 1 to 2. As you may have observed in the older mechanical odometers when 99999 turns over to 100000 the mechanism struggles since the rule needs to be applied in sequence five times: first the "1"'s digit goes from 9 to 0, this triggers the increment of the "10"'s digit but this then goes from 9 to 0 so triggering the "100"'s digit etc. In some older odometers the "100000"'s digit has not done changing when the "1"s digit is already moved one from 0 to 1. The same issue occurs in the binary electronic implementation.

[0036] Figure 10 is a block diagram of a ripple carry counter. The clock to the next higher significance bit (the MSB is on the right) is simply the negative edge of the lesser significance bit. This technique works, but the clocks are delayed later and later in time as the bits "roll over" – similar to the odometer example.

[0037] Figure 11 is a block diagram of a ripple carry counter clocking the elements on the same clock edge. A further desirable feature of this circuit to make a binary counter is that the clocks are all common in the DTypes. This aspect takes advantage of software tools that check timing when all the clocks are connected in common. Consequently, some design methodologies enforce the rule that all clocks must be connected together. Now all the DTypes are clocked on the same clock edge, there is no skew, and each Dtype will toggle if the upper input to the Xor gate is a one. Hence, each bit will toggle if the bit below it is a "1" and it is about to toggle. Although there is now no skew, the time to propagate the signal to toggle via the AND gates increases with the length of the counter (i.e. the bit width) and this will limit the speed of the counter. This connection scheme clarifies the advantage of the aforementioned look-ahead carry generator: a look-ahead carry or parallel carry generator, simply removes the chain of AND gates and replaces it with un-chained combinatorial gates.

[0038] Figure 12 is a block diagram of a ripple carry counter, clocking the elements on the same clock edge, which stops the carry in every bit cell. Here the carry is stopped in every bit cell - it does not propagate for more than the AND gate (and through the XOR) but the count sequence is not greatly modified – it is still close to binary - the high significance bit transitions are always one clock cycle later than they would be in a pure binary count. We have just used the synchronous negative edge detector. In place of allowing the negative edge itself to cause an event, we have a two stage pipeline and we ask if the first element is 0 and the second 1.

[0039] Figure 13 is a table listing the count sequence of the "S"-state DTypes in Figure 12. The LSB alternates each cycle; the next LSB alternates every two cycles and so forth, as in the usual binary count, but the MSB transitions are delayed by one cycle for each bit. The states of the "C" DTypes are simply a one-clock cycle delayed version of the "S" state.

[0040] This method detects the negative edge by asking if the old state of any particular bit is a 1 and the new state is a 0. The count sequence it creates is easily understood. How we can find the "index" of the state? We see from the table on the last page that state 4 is actually 00010 which in binary would be 2, and state 6 is 00100 which in binary is 4, so clearly we can't simply use the count code as though it were binary.

[0041] Figure 14 is a block diagram of a ripple counter with "C"-state DTypes which indicate whether the following "S"-state DType should toggle on the next clock edge, representing an improvement upon the block diagram in Figure 12 due to the re-positioning of the AND gate such that only one gate delay (of AND or Exclusive-Or) is interposed between each DType.

[0042] Note that the C states are now not a simple delayed copy of the S states: a single AND gate lies between them. The C state is now a bit indicating that the next S state should toggle on the next clock.

[0043] Figure 15 is a table listing the count sequence of the "C"-state and "S"-state DTypes in Figure 14. The bits for a 6 bit implementation are shown in binary. The left column is the state of the "C" DTypes, the center column is the state of the "S" DTypes and the final column is a calculation of the state index. When the logic is modified the summation of S + 2C is a linear representation of the state. In fact the calculation of S + 2C results in the normal binary state index plus the number of bits in the counter (6 in this case). We need only present the C and S bus to an adder and form the summation to get the index. Furthermore, using modulo arithmetic to determine how many counts have occurred, we can ignore the addition of the number of bits (i.e. ignore the +6 in this case). Specifically, if the rate of a signal needs to be determined we can apply that signal as the clock to this modified counter: the counter will allow a clock of over

1GHz to be counted. To determine the number of cycles that have passed, we do not reset this modified counter, we simply snapshot the state into a register at the reference time. For example, in this six bit case discussed here we snapshot the 12 Dtype states. We form the summation of S and 2C modulo 64 (ie in a six bit field). Then the next time the reference edge comes along we snapshot again. Again we calculate the summation modulo 64 and take the difference modulo 64 – the result is the number of cycles that have occurred between the two snapshots.

[0044] The following Verilog code describes an example of the oscillator, (the variable "Osc") the two point sampling mechanism (unsynchronized "ASYNPHASE" and synchronized "SYNCPHASE") and the "almost binary" counter (S elements are "S" C elements are "C").

[0045] Various delays in (ie uses of #12345 etc) are provided as an illustrative realistic example of the potential delay when simulating this code.

[0046]

```
`timescale 10fs / 10fs
module OSCCNTR(REFCLK, CLKOSC, SYNCPHASE, ASYNPHASE, CD, SD);
   parameter WIDTH = 12;    // This is the width of the Counter
section
   parameter TGATE = 11111;// Nominal time delay of each gate in
                           //  1/100th pico-seconds. The
oscillator runs
                           //  at a rate that is 1/(10*TGATE)
   // 11111 => 909MHz
   // 5882  => 1.7G
   parameter JITTER = 15;  // RMS Jitter in 1/100th pico-seconds
on the
                           //  oscillator gate delay (note: on
the
                           //  gate delay - the noise on the
                           //  oscillator is larger than this)
   // I am going to try to model some phase noise in the
oscillator. I
   //  am using the common "hack" that the sum of 12 rectangular
random
   //  numbers is about Gaussian and I will use the correlation
```

```
   //  coefficient CC like this:
   //       Next_Random = CC.Last_Random + sqrt(1-CC^2).Random()
   // These next two parameters are a CC of 0.9995 and CC2 is
actually
   // just sqrt(1-0.9995^2)
   parameter CC = 0.9995;   // Correlation coefficient of the
gate
   parameter CC2 = 0.03162; //  delay - the closer this is to 1
the
                            //  more low frequency variation is
                            //  present in the oscillator.
   parameter MI = WIDTH - 1;

//=============================================================
====
   // This first section just sets up a function that will give
us the
   // noise of the oscillator...
   function integer osc_dly (input integer last_delay);
      begin
         osc_dly = CC*last_delay
                   + CC2 * JITTER *
                   (  $random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000
                     +$random()%5000 ) / 10000;
```

```
      end
   endfunction // osc_dly

//=================================================================
====
   // Now the IO port definitions. Note that although the
Counter
   // width is parameterized (by WIDTH) the oscillator width is
hard
   // coded... (too difficult to allow it to be a variable!)
   output CLKOSC;    // The high speed output clock.
   input  REFCLK;    // The input clock from the Reference
(typically
                     // this will be the 27MHz reference divided
by 16)
   output [14:0] ASYNPHASE;    // Phase of the Oscillator as
sampled by the
   reg    [14:0] ASYNPHASE;    //  REFCLK clock
   output [14:0] SYNCPHASE;    // Phase of the Oscillator as
sampled by the
   reg    [14:0] SYNCPHASE;    //  ClkSync - that is the REFCLK
                               //  synchronized into the CLKOSC
domain
      output [MI:0]   SD;   // S and C registers as defined in
   output [MI-1:0] CD;    //  for the "Almost Binary" counter.
   reg [MI-1:0] C, CD;    //  (basically the S register is the
state, the
   reg [MI:0]   S, SD;    //   C register is the pipelined carry.
SD is
                          //   the sampled S and CD is the
sampled C)
   // Internal wire/register definitions:
   reg        ClkSync;    // Sampled version of the REFCLK in
CLKOSC
```

```
   reg        ClkSync2;  // Second sample of above used for edge
                         //  detect
      reg [14:0] Osc;       // The actual oscillator itself..

//=================================================================
====
   // The oscillator:
   integer    osctime;   // This is the time delay of the
oscillator
                         // gate itself and varies each
oscillator
   initial               // cycle ...
     begin
        osctime = TGATE;
        Osc = 15'b010101010101010;
        #10 Osc[0] = 1;
     end
   always @(Osc[0])  Osc[1]  <= #osctime ~Osc[0];
   always @(Osc[1])  Osc[2]  <= #osctime ~Osc[1];
   always @(Osc[2])  Osc[3]  <= #osctime ~Osc[2];
   always @(Osc[3])  Osc[4]  <= #osctime ~Osc[3];
   always @(Osc[4])  Osc[5]  <= #osctime ~Osc[4];
   always @(Osc[5])  Osc[6]  <= #osctime ~Osc[5];
   always @(Osc[6])  Osc[7]  <= #osctime ~Osc[6];
   always @(Osc[7])  Osc[8]  <= #osctime ~Osc[7];
   always @(Osc[8])  Osc[9]  <= #osctime ~Osc[8];
   always @(Osc[9])  Osc[10] <= #osctime ~Osc[9];
   always @(Osc[10]) Osc[11] <= #osctime ~Osc[10];
   always @(Osc[11]) Osc[12] <= #osctime ~Osc[11];
   always @(Osc[12]) Osc[13] <= #osctime ~Osc[12];
   always @(Osc[13]) Osc[14] <= #osctime ~Osc[13];
   always @(Osc[14]) Osc[0]  <= #osctime ~Osc[14];
   // Now although there are 15 oscillator sections we still
want to
```

```
   // derive the clock at the rate of every 5....
   assign #12000 CLKOSC = (Osc[0]  & Osc[5] )
                        | (Osc[5]  & Osc[10])
                        | (Osc[10] & Osc[0] );
   // Each time the CLKOSC makes an output, allow the time to
shift a
   // little - this models the jitter on the clock...
   always @(posedge CLKOSC) osctime <= TGATE + osc_dly(osctime-
TGATE);

//================================================================
====
   // Now the two registers that snapshot the oscillator state:
   // This is the asynchronous one:
   always @(posedge REFCLK) begin ASYNPHASE <= Osc; end
   // and this is the synchronous one:
   always @(posedge CLKOSC)
     begin
        ClkSync2 <= ClkSync;
        ClkSync  <= REFCLK;
        if (ClkSync & ~ClkSync2)
          begin
             SYNCPHASE <= Osc;
          end
     end

//================================================================
====
   // Now the pipelined carry counter that accumulates the count
between
   // the samples... (this will be post-processed by synthesized
code
   // that finds the actual cycle difference). This counter just
runs
```

```
   // all the time on the CLKOSC clock. I have chosen to use a
12 bit
   // wide implementation.
      integer      i;
   always @(posedge CLKOSC)
     begin
        // This first section is the counter:
        for (i = 1; i < WIDTH; i = i + 1)
          if (C[i-1]) S[i] <= #14100 ~S[i];
        // This second section is the carry state:
        for (i = 1; i < MI;    i = i + 1)
          C[i] <= #18800 C[i-1] & S[i];
        // These are the two "end cases"
        S[0] <= #13500 ~S[0];
        C[0] <= #12100  S[0];
        // This is the sample of the counter taken by the
REFCLK:
        if (ClkSync & ~ClkSync2)
          begin
             CD <= #9300 C;
             SD <= #8900 S;
          end
             end // always @ (posedge CLKOSC)
endmodule // OSCCNTR
```

[0047] The following discusses calculating the elapsed cycles of the oscillator.

[0048] The above description of the two-point sampling of the three section oscillator and the "almost binary" counter (together with the Verilog code example), result in the presentation of four variables to the succeeding logic controller: namely, on each event of the REFCLK (the external known frequency reference) ASYNPHASE, SYNCPHASE, CD and SD. These variables represent respectively the phase of the oscillator at the moment of the external event; the phase of the oscillator when the event is first registered into the free running oscillator clock domain, the state of "C" registers (the carry registers) in the "almost binary" counter and the state of the "S" register (the state registers) in the "almost binary" counter. From these four

variables it is now possible to determined, without fault due to any timing relationships, the state of the oscillator to with 1/10th of one cycle and consequently the fractional number of cycles of the oscillator between the REFCLK events. The calculation to achieve this includes two parts, the logic to determine the phase difference called "DPhase" and described below; and the logic to measure the frequency called "Fmeas" in the following pages. (Again, all reference to time intervals in this description are illustrative examples of the simulation forcing a realistic delay in the simulation).

[0049]

```
`timescale 1ps / 1ps
module DPhase (SyncPhase, AsynPhase, DelPhase);
   input [14:0] AsynPhase;  // AsynPhase is the 15 element
oscillator
                            //  sampled by the asynchronous edge
of the
                            //  reference clock..
   input [14:0] SyncPhase;  // SyncPhase is the 15 element
oscillator
                            //  sampled by the synchronous edge
of the
                            //  reference clock..
   output [5:0] DelPhase;   // The Difference in phase
accounting for
                            //  roll over etc
      wire signed [4:0] SyncCount,  AsynCount;
   wire        [3:0] SyncCountU, AsynCountU;
   wire signed [5:0] DelPhase1;
         // These next two assigns calculate the SyncCount
value...
   assign #5234  SyncCountU
          = SyncPhase[0]  + !SyncPhase[1]
          + SyncPhase[2]  + !SyncPhase[3]
          + SyncPhase[4]  + !SyncPhase[5]
          + SyncPhase[6]  + !SyncPhase[7]
          + SyncPhase[8]  + !SyncPhase[9]
```

```
            + SyncPhase[10] + !SyncPhase[11]
            + SyncPhase[12] + !SyncPhase[13]
            + SyncPhase[14];
      assign #500 SyncCount = (SyncPhase[0] == 1)
                            ?  SyncCountU
                            : -SyncCountU;
         // These next two assigns calculate the AsynCount value...
      assign #5234 AsynCountU
            = AsynPhase[0]  + !AsynPhase[1]
            + AsynPhase[2]  + !AsynPhase[3]
            + AsynPhase[4]  + !AsynPhase[5]
            + AsynPhase[6]  + !AsynPhase[7]
            + AsynPhase[8]  + !AsynPhase[9]
            + AsynPhase[10] + !AsynPhase[11]
            + AsynPhase[12] + !AsynPhase[13]
            + AsynPhase[14];
      assign #500 AsynCount = (AsynPhase[0] == 1)
                             ?  AsynCountU
                             : -AsynCountU;
      assign #1987 DelPhase1 = AsynCount - SyncCount;
      assign #500 DelPhase = (DelPhase1 > 0)
                           ? (DelPhase1 - 30)
                           :  DelPhase1;
   endmodule // DPhase
```

[0050] Note that the module above has assigned SyncCountU and AsynCountU making use of the fact that each oscillator element is inverted relative to the adjacent ones. The phase difference (DelPhase) is then simply the difference of the signed version of the SyncCountU and AsynCountU (namely AsynCount and SyncCount) reduced to modulo 30 (since there are 30 states of the 15 elements).

[0051] The Fmeas unit itself then uses the "C" and "S" states as sampled by the external event (ie the SD and CD variables) and together with a call to the above module 9the instance DP in the code below) calculates the Count number – this is the number of 1/10th cycles of the free running oscillator that have passed between the REFCLK events.

[0052]

```
`timescale 1ps / 1ps
module Fmeas (RefClk, ClkEn, SyncPhase, AsynPhase, CD, SD,
Count);
   parameter WIDTH = 12;   // This is the width of the Counter
section
                           //  as used in the GDS cell.
Currently
                           //  assumed to be 12
   parameter MI = WIDTH - 1;
      input  RefClk;   // The input clock from the Reference -
about 27Mhz
   input  ClkEn;    // Qualifies RefClk to about /4 or /16
   input [14:0] AsynPhase;   // Phase of the Oscillator as
sampled by the
                             //  RefClk clock
   input [14:0] SyncPhase;   // Phase of the Oscillator as
sampled by the
                             //  ClkSync - that is the RefClk
                             //  synchronized into the ClkOsc
domain
      input [MI:0]   SD;   // S and C registers as defined in
   input [MI-1:0] CD;   //
"file://capella/e/design/GenericNotes/PCC.doc"
   output signed [MI+4:0] Count; // The actual output frequency
as measured
   wire signed [MI:0] Count0;  // The integer (none-fractional)
count
   wire [MI:0]   Count1;       // The first sum output of the CD
an SD
   reg [MI:0]    Count2;       // The last value of Count1
   assign #5444  Count0 = Count1 - Count2;
   assign #5333  Count1 = SD + 2*CD;
   wire signed [5:0]     DelPhase0;
```

```
wire        [5:0]     DelPhase1;  // Fraction of one cycle..
reg         [5:0]     DelPhase2;
assign #4565  DelPhase0 = DelPhase1 - DelPhase2;
DPhase DP(SyncPhase, AsynPhase, DelPhase1);
 always @(posedge RefClk)
  if (ClkEn)
    begin
       Count2    <= #233 Count1;
       DelPhase2 <= #233 DelPhase1;
    end
assign        Count = 10*Count0 + DelPhase0;
endmodule // Fmeas
```

[0053] The following discusses the frequency correction logic.

[0054] The above sections have described how the logic is able to determine how many 1/10ths of cycles of the oscillator, which runs at about 1.3Ghz have passed. Effectively it is determining the frequency of an equivalent 13Ghz oscillator. Logic uses this frequency measurement to correct the data going to the DFG core (to that section of the DFG that actually interpolates between the edges of the free running oscillator). An interpolator (the block named DFG INTER) outputs a frequency Fout = FOSC/Control. If the FOSC (which is the free running oscillator) frequency increases, the Control variable input must increase as well. Since Fmeas as described is generating a number Count which is equal to FOSC/FREF if we arrange that Control = Count*FREQ (ie that the control variable to the DFG INTER is simply equal to the product of the Count variable from Fmeas and the requested frequency input) then we see that Fosc cancels out and we are left with Fout = FREF/FREQ. We have an output frequency controlled by FREQ and independent of the free running oscillator. There is no feedback involved, and the rate of measurement of the oscillator can be arbitrarily high. In one embodiment, the Fmeas unit runs at 1.6Mhz (so suppressing any phase noise at a frequency below 1.6Mhz) and the FREQ bus is a 20 bit wide bus (thereby giving a high resolution set point frequency).

[0055] The following is a discussion of various particular embodiments.

[0056] One embodiment is a feed-forward control loop circuit, which generates a fixed and predictable output frequency from a possibly variable and unpredictable source of relatively high frequency oscillations. Included is an oscillation source of relatively high frequency and potentially unpredictable and unstable events. Included is a reference frequency source of

potentially lower frequency, unrelated to the oscillation source, from which a timing reference is derived. Included is a measurement unit, such as "fmeas", responsive to the reference source and the oscillation source, operating to assess the frequency of the oscillation source relative to the reference source. Included is an interpolating device, or frequency changing device, such as described in US Patent Application Publication No. 20080285698, hereby incorporated by reference, capable of accepting one frequency as input and creating a second frequency as output dependent on a second input. The first input of this device is connected to the oscillation source. Included is a unit operating on the output of the measurement unit and receiving a requested frequency input, controlling the interpolator or frequency changing device on its second input, such that a feed-forward control loop is created producing an output from the interpolator dependent only on the reference source and the requested frequency input.

[0057] On embodiment is an "almost-binary" counter. A high speed counting device produces a sequence of states in response to a clock input, such that the number of states between any two states can be conveniently calculated. Included are one or more state-variables ("C" or "Carry" DTypes) configured to sample an input and synchronously detect a change on that input as a function of the difference between the input and the input as sampled, and to provide this detection of change to an output. Included are two or more state-variables ("S" or "State" DTypes) configured to change state or not change state as determined by a controlling input. Included are connections between the C DTypes and S DTypes such that the detection of change output is connected to the controlling input and consequently causes change in the output when a change in input is detected.

[0058] Another embodiment is the "almost-binary" counter which implements up-count. The change on the input is from a "one" to a "zero" state, so causing a change to the associated S state variable when a negative edge is synchronously detected. An example is shown in Figure 12.

[0059] Another embodiment is the "almost-binary" counter, optimized. A high speed counting device produces a sequence of states in response to a clock input, such that the number of states between any two states can be conveniently calculated. Included are two or more state variables ("S" or "State" DTypes) and associated "ripple-through" carry logic creating an enabling signal output to allow more significant bits to change when the local S bit is in the appropriate state, and the enable signal input is indicating that all lower significance bits are also in the state to allow a carry output. Included are one or more state variables ("C" or "Carry" DTypes) interposed between the enabling signal output of the lower significance bits and the

enable signal input of the more significant bits. Thereby created is at least one clock cycle delay in the propagation of the carry from less to more significant bits.

[0060] Another embodiment is the "almost-binary" counter which creates up-count. The appropriate state for generation of carry output is a "one" thereby creating an "up-counter". An example is shown in Figure 14.

[0061] Another embodiment is the "almost-binary" counter which calculates the difference between successive states. Included are a set of latches that sample the value of the S and C DTypes in response to an external, possibly asynchronous, signal. The latches providing a set of staticized signals S' and C' correspond to the value of the S and C registers at the time of the external input. Included is a combinatorial logic element that calculates the quantity S+2C-S'-2C' thereby generating a number indicative of the number of cycles of the counter between the present state (S, C) and the state as sampled (S',C')

[0062] Another embodiment is the "almost-binary" counter with modulo operation. The finite number of S elements in the device is ignored, and the combinatorial element is allowed to operate in the same finite word width, thereby forcing all arithmetic to be modulo two to the power of the finite number of elements.

[0063] Another embodiment is a longer-than-necessary ring oscillator to resolve sampling timing error, with an oscillating series of delay elements. Included are a plurality of elements connected end to end in a ring and arranged such that an oscillating pattern is created in the ring – a so-called "Ring Oscillator". Generated is an output pulse at a rate greater than the rate of oscillation of any given element in the ring, specifically at a rate equal to N times the rate of oscillation of any given ring element, where N is an integer greater than one, thereby generating a output that repeats N times within one complete cycle of the ring. Sampled, in response to an external, possibly asynchronous, event, is the state of the elements of the ring thereby generating a first sample of the ring called the "asynphase" of the ring. Sampled is the external event into a state variable clocked by the output pulse. Sampled, in response to the appearance of a sampled event in the state variable, is the state of the elements of the ring thereby generating a second sample of the ring called the "syncphase" of the ring.

[0064] Another embodiment is a ring oscillator and counter to determine event time. Included is a ring with at least two interconnected elements, configured to oscillate and thereby generate output pulses at a certain rate representative of the time delay of at least two of the elements of the ring. Sampled is the state of the elements of the ring upon the appearance of an external, possibly asynchronous, event, thereby creating the "oscillator phase sample". Included is a high speed counter. Sampled is the state of the high speed counter upon appearance of a

sample in a state variable clocked by the pulses from the ring oscillator and representing a staticized, possibly asynchronous, external event, thereby creating a sample of the counter state called the "counter sample". Input are the current value of the "oscillator phase sample" and the "counter sample" and at least one historical record of the same values, and this operates to determine the difference in count in the successive "counter samples" and the fractional difference in oscillator phase from the successive samples of the "oscillator phase sample" values, thereby creating a measurement of time interval at a resolution in time smaller than the time interval of the high speed counter.

[0065] Another embodiment is the error-free ring oscillator and counter to determine event time. Included is the ring oscillator as described. Included is a high speed counter. Sampled is the state of the high speed counter upon appearance of a sample in a state variable clocked by the pulses from the ring oscillator and representing a staticized, possibly asynchronous, external event, thereby creating a sample of the counter state called the "counter sample". Input are the current value of the "asynphase" and "syncphase" of the oscillator and the "counter sample" and at least one historical record of the same values, and this operates to determine the difference in count in the successive "counter samples" and the fractional difference in oscillator phase from the successive samples of the "asynphase" and "syncphase" values, thereby creating a measurement of time interval to a resolution in time smaller than the time interval of the high speed counter.

[0066] Another embodiment is a ring oscillator and the almost-binary counter that determine event time. Included is a ring with at least two interconnected elements, configured to oscillate and thereby generate output pulses at a certain rate representative of the time delay of at least two of the elements of the ring. Sampled is the state of the elements of the ring upon the appearance of an external, possibly asynchronous, event, thereby creating the "oscillator phase sample". Included is an "almost-binary" counter as described. Operated upon are at least two successive samples of the "oscillator phase sample" and the output of the modulo arithmetic element as described, configured to create a measurement of time interval to a resolution in time smaller than the time interval of the high speed counter.

[0067] Another embodiment is the error-free ring oscillator and the almost-binary counter to determine event time. Included is a ring oscillator as described. Included is an "almost-binary" counter as described. Input are current value of the "asynphase" and "syncphase" of the oscillator and the output of the modulo arithmetic element as described, and at least one historical record of the same values, and this operates to determine the difference in count in the successive "counter samples" and the fractional difference in oscillator phase from the

successive samples of the "asynphase" and "syncphase" values, thereby creating a measurement of time interval to a resolution in time smaller than the time interval of the high speed counter.

[0068] Another embodiment is a feed-forward control loop using the error-free ring oscillator and almost-binary counter. Generated is a fixed and predictable output frequency from a possibly variable and unpredictable source of relatively high frequency oscillations. Included is a feed-forward control loop as described. Included are a ring oscillator and the almost-binary counter as described.

[0069] While the present invention is disclosed by reference to the preferred embodiments and examples detailed above, it is to be understood that these examples are intended in an illustrative rather than in a limiting sense. It is contemplated that modifications and combinations will readily occur to those skilled in the art, which modifications and combinations will be within the spirit of the invention and the scope of the following claims. What is claimed is:

CLAIMS

1. A feed-forward control loop circuit, comprising:

an oscillation source having a first frequency;

a reference frequency source having a second frequency independently generated from the oscillation source, wherein a timing reference is based on the reference frequency source;

a measurement circuit responsive to the reference source and the oscillation source, the measurement unit assessing the first frequency relative to the second frequency;

a first circuit accepting an input frequency and creating an output frequency dependent on another input, wherein the input frequency is determined by the oscillation source; and

a second circuit operating on an output of the measurement circuit and receiving a frequency input, the second circuit controlling a second input of the first circuit, such that a feed-forward control loop produces an output from the first circuit, the reference source and the frequency input being sufficient to determine the output of the feed-forward control loop.

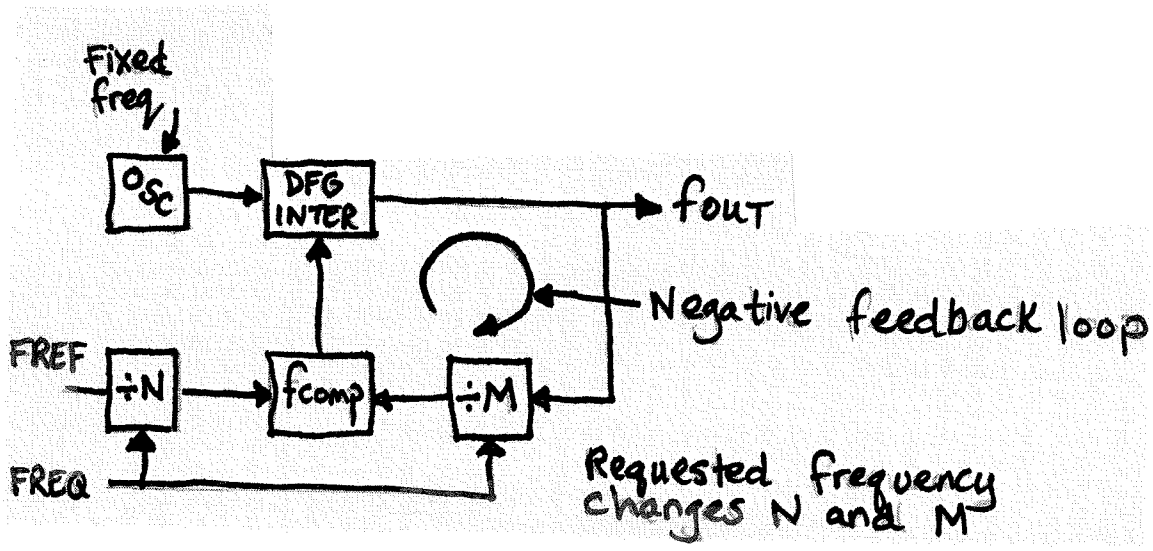
Fixed freq
OSC
DFG INTER
fOUT
Negative feedback loop
FREF
÷N
fcomp
÷M
FREQ
Requested frequency changes N and M

Figure 1

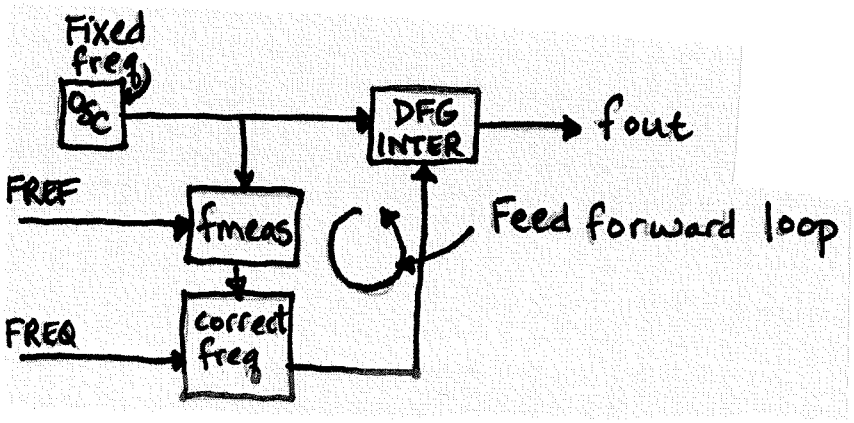
Fixed freq
OSC
DFG INTER
fout
FREF
fmeas
Feed forward loop
FREQ
correct freq

Figure 2

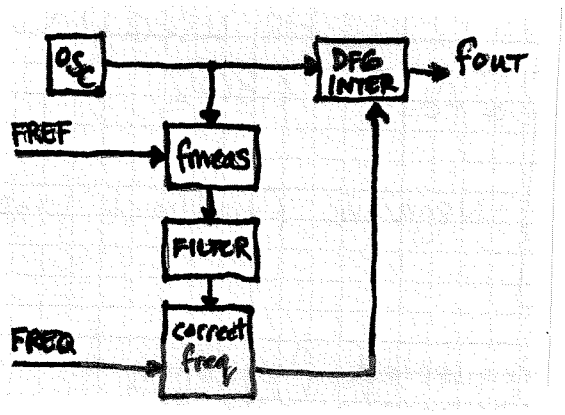

Figure 3

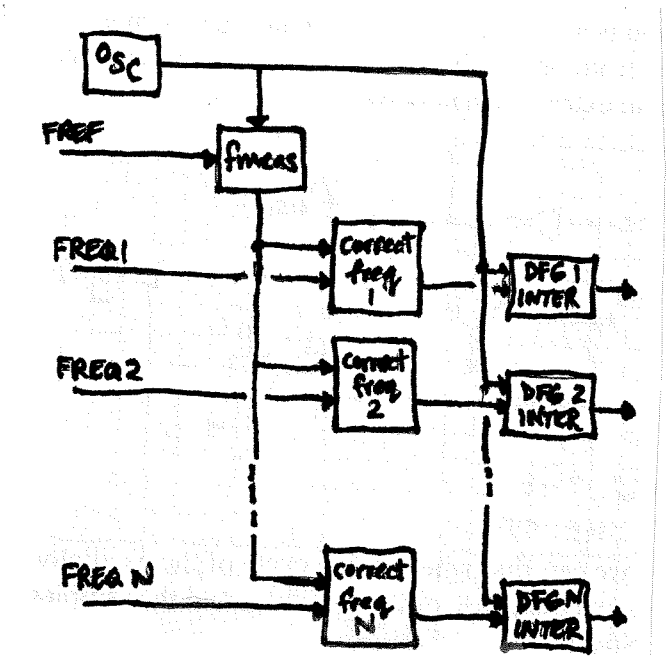
OSC
FREF
fmeas
FREQ1
Correct freq 1
DFG 1 INTER
FREQ2
Correct freq 2
DFG 2 INTER
FREQ N
correct freq N
DFG N INTER

Figure 4

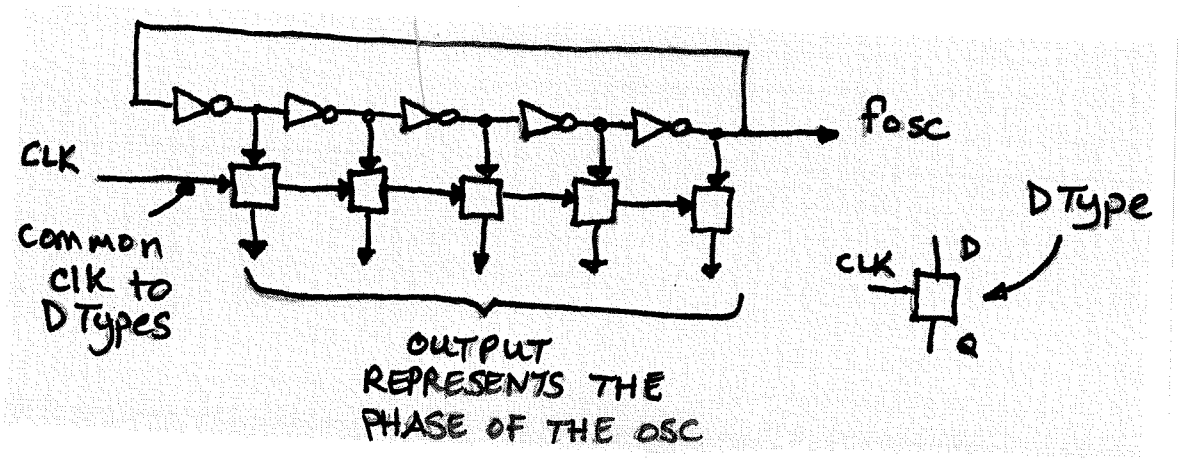
fosc
CLK
common
clk to
D Types
OUTPUT
REPRESENTS THE
PHASE OF THE OSC
DType
CLK
D
Q

Figure 5

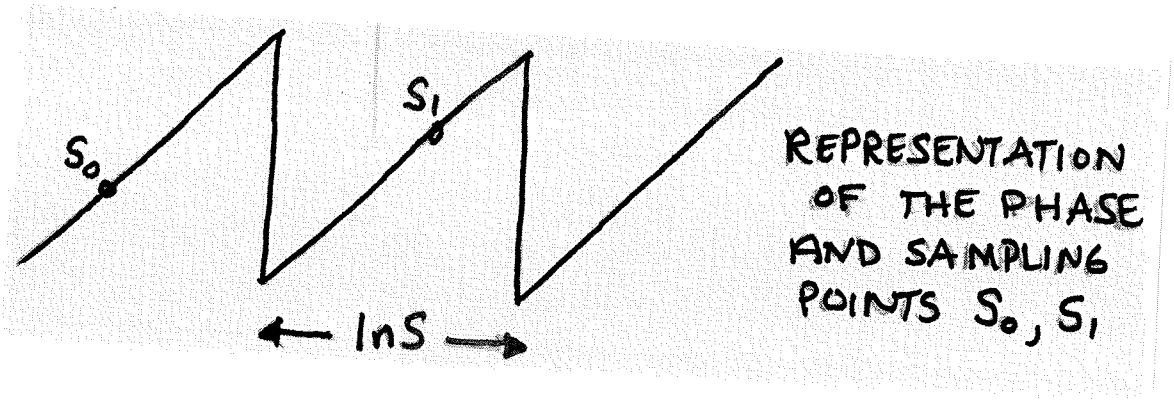
S_0
S_1
InS
REPRESENTATION OF THE PHASE AND SAMPLING POINTS S_0, S_1

Figure 6

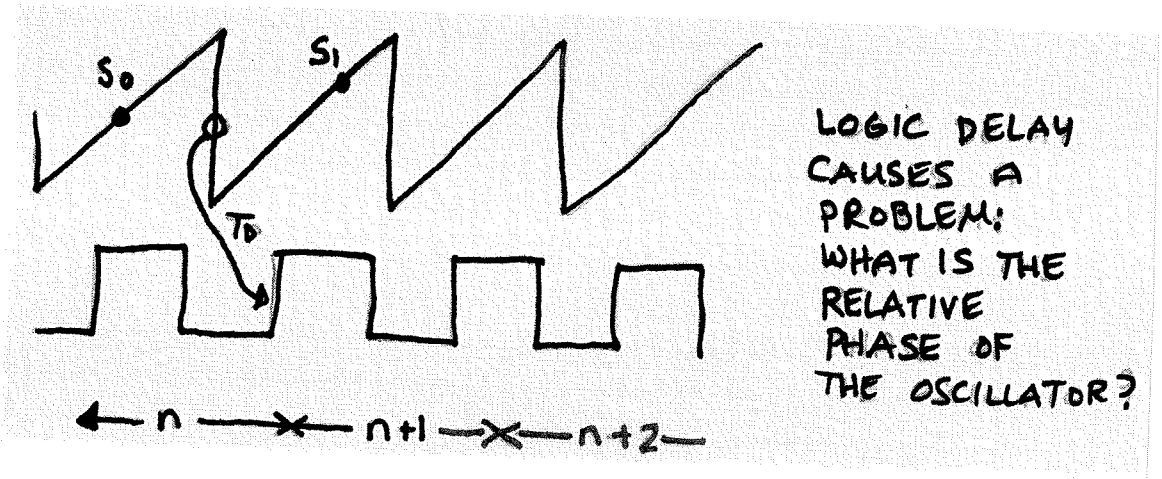
S0
S1
TD
n
n+1
n+2
LOGIC DELAY CAUSES A PROBLEM: WHAT IS THE RELATIVE PHASE OF THE OSCILLATOR?

Figure 7

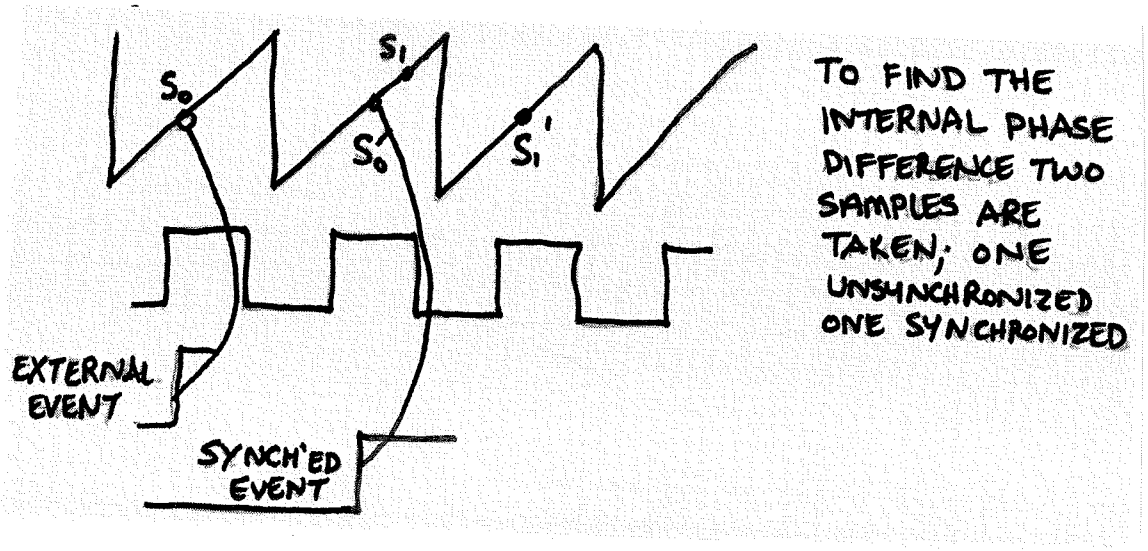
S_0
S_1
S_0'
S_1'
EXTERNAL EVENT
SYNCH'ED EVENT
TO FIND THE INTERNAL PHASE DIFFERENCE TWO SAMPLES ARE TAKEN; ONE UNSYNCHRONIZED ONE SYNCHRONIZED

Figure 8

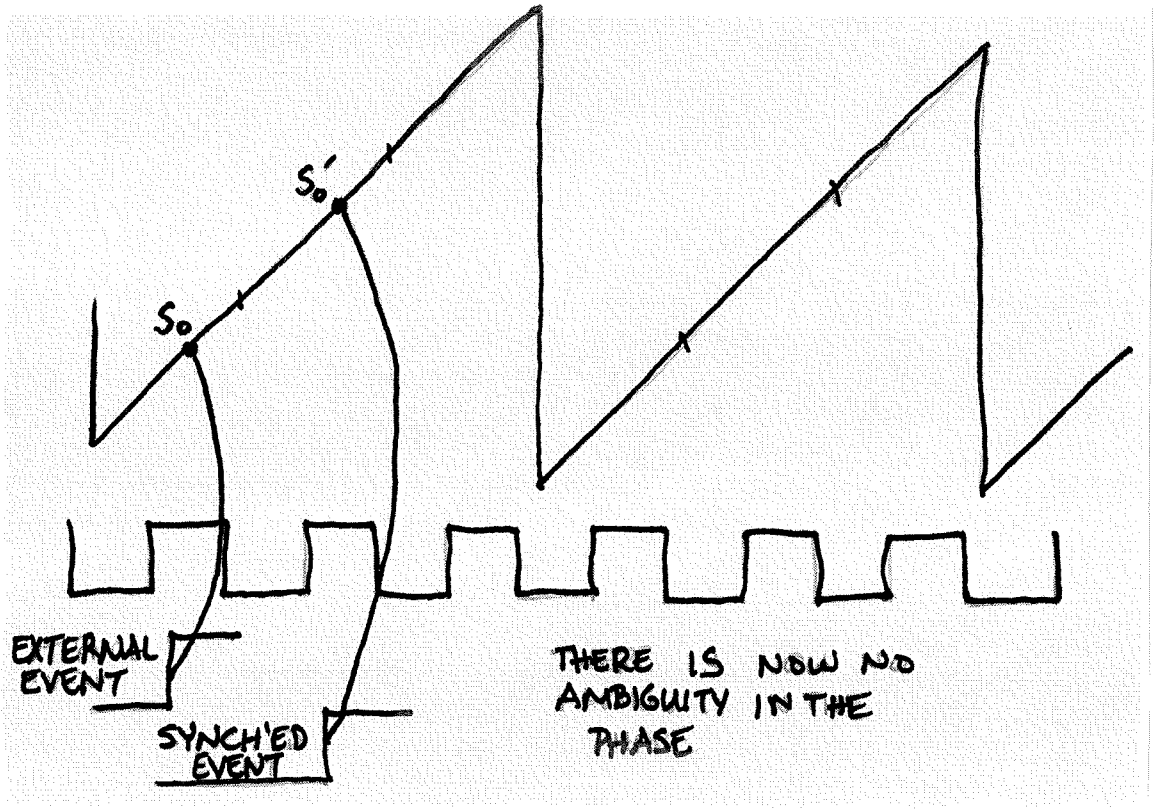
S₀'
S₀
EXTERNAL EVENT
SYNCH'ED EVENT
THERE IS NOW NO AMBIGUITY IN THE PHASE

Figure 9

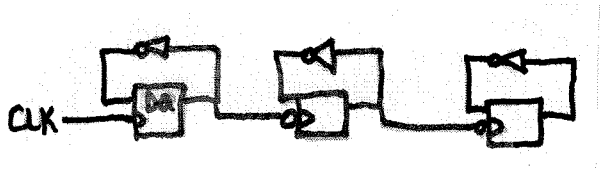

Figure 10

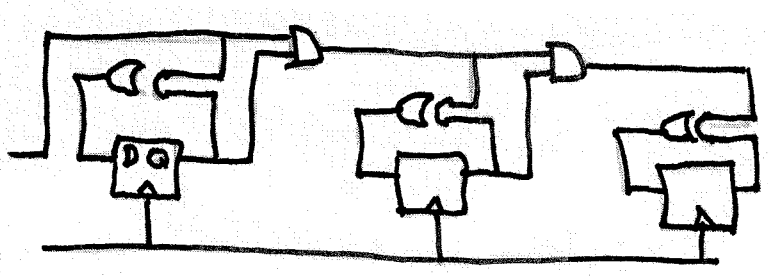

Figure 11

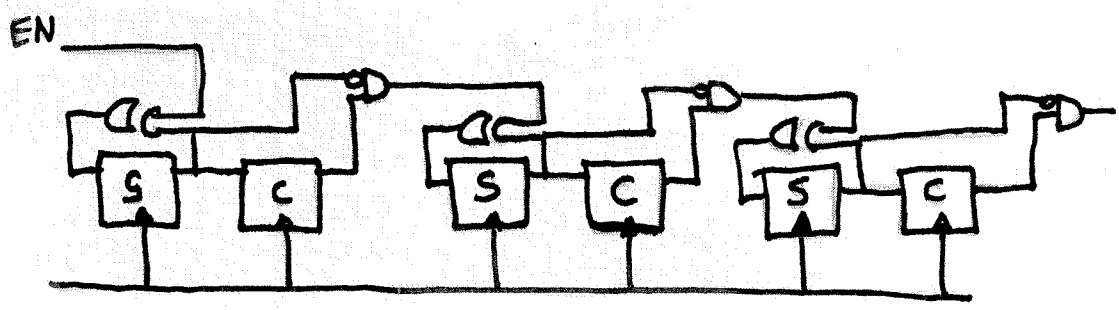

Figure 12

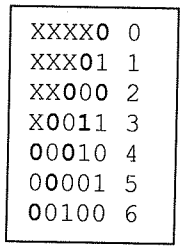

Figure 13

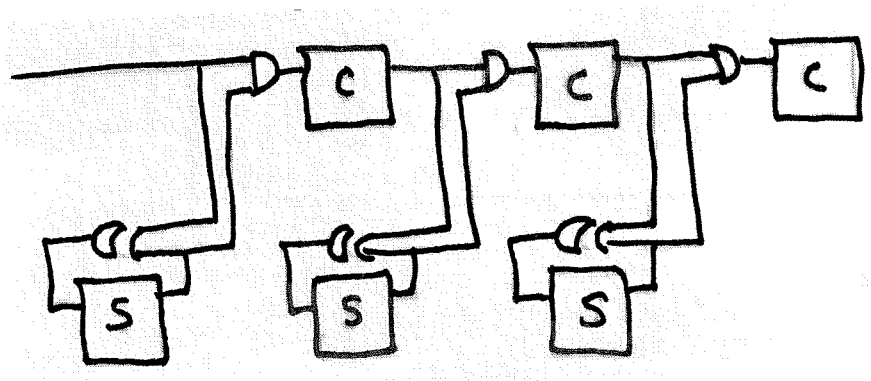

Figure 14

C	S	Y
000000	111111	63
000001	111110	64
000010	111101	65
000101	111000	66
001000	110011	67
010001	100010	68
100010	000001	69
000001	000100	6
000000	000111	7
000001	000110	8
000010	000101	9
000101	000000	10
000000	001011	11
000001	001010	12
000010	001001	13
000001	001100	14
.	. . .	
000000	111111	63
000001	111110	64
000010	111101	65
000101	111000	66
001000	110011	67
010001	100010	68
100010	000001	69
000001	000100	6
000000	000111	7
000001	000110	8
000010	000101	9

Figure 15