Low cost, high performance frequency/interval counters

Kasper Pedersen
kkp2008@kasperkp.dk

Abstract

Frequency counters can be designed with good, bad, or ugly performance. For some reason bad and ugly are most popular. We describe a way to build a simple counter that has better than 100 ps RMS noise while maintaining low cost, and only uses easily obtainable components.

Introduction

The simplest approach to a frequency counter is the gated counter. The signal of interest is fed to a counter which in turn is gated by a pulse of known length, such as 1 second. The shortcoming of this approach is that the quantiztion error is 1/2 LSB regardless of input frequency, so with a 1 second gate the quantization error is 0.5Hz. To get precision on the order of 1ppb on a 10MHz input, the gate time would have to be 100 seconds. On a 30Hz signal the gate time would have to be one year.

A slightly better approach is the reciprocal counter. Instead of counting the signal gated by the reference, the reference is counted and gated by the signal, divided down so the gate time is, say, one second. Such a counter with a 10MHz reference will have 10 million counts per second regardless of input, and will need 100 seconds gate time to achieve 1ppb precision regardless of input frequency. The simple way to get better performance is to increase the reference frequency, only 10MHz is the industry standard for frequency standards, the PLLs required for upconversion add cost, and beyond 200MHz there are no cheap prototyping-friendly CPLDs.

The reciprocal counter can be greatly improved at little cost by adding an interpolator. The interpolator converts the quantization error of the counter circuit into an analog value, converts the analog value to digital and treats it as a fractional count.

The counter described herein uses a low cost interpolator, consisting of a CPLD, a few diodes, and the A/D converter in a microcontroller.

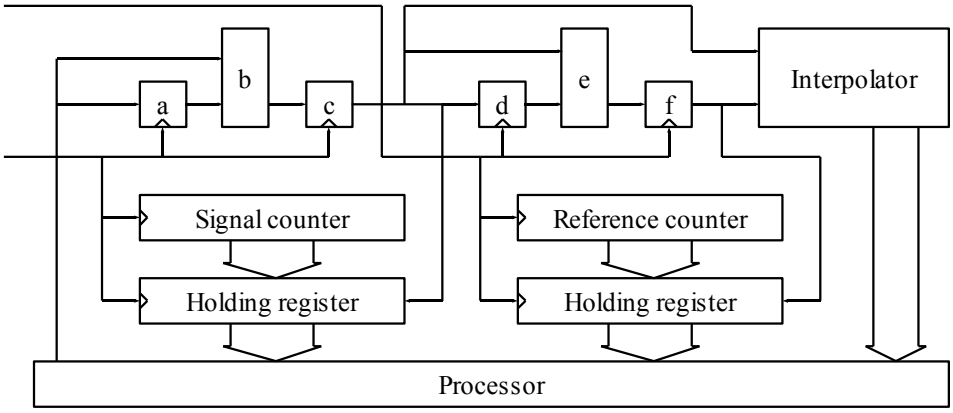


fig.1

The reciprocal interpolating counter

The signal clock is applied to the clock input of synchronizer flipflop *a*, and a control term to the D input of flipflop *a*. When the host processor desires to timestamp an edge, it sets the control term low. One or two signal clocks later, depending on multiplexer *b*, the holding register for the signal counter will be disabled, capturing the value of the signal counter.
The signal from flipflop *c* then crosses clock domains to the second part of the counter. Synchronizer flipflops *d* and *f* similarly disable the holding register on the reference counter side. The two holding registers now hold the signal edge number and the reference edge number. A reciprocal frequency counter triggers this process twice, subtracts*, and divides the two differences to yield the ratio between the signal and reference.

The quantization error occurs at the input of flipflop *d* (or *f* depending on multiplexer *e*) where the signal crosses clock domains. Thus, the interpolator will see a time difference between its two inputs of 0-1 clocks or 1-2 clocks depending on multiplexer e, representing the instantaneous phase of the reference clock relative to the signal edge captured. This difference must be converted to a digital value with high precision and linearity, and directly limits the attainable precision for the whole system. A system without interpolator has an inherent 1 clock limit.

*usually by clearing both counters on the first trigger

Digital section

The goal is to keep cost low. Since the reference clock chosen is 10 or 20MHz this means we can clock the host microprocessor from the reference. The gain is twofold: We eliminate a separate crystal for the cpu, and even the cheapest members of ATMega series have a 16 bit timer/counter with input capture capability, providing the reference counter directly on the cpu. It needs to be lenghtened to 32 bits or more, this can be done in software using an overflow interrupt.

The signal counter needs to run at reasonably high frequencies though this can be helped by a prescaler. The effective length of the signal counter needs to be greater than the maximum capture interval, a million counts or more at high frequencies and one second interval. Like the reference counter it can be lengthened in software by observing the rollover of the counter, in practice the falling edge of the topmost bit. Thus, 16 bits in hardware is plenty for the signal counter and the holding register.

Synchronizers *c,f* need low phase noise since noise here adds to the output. A MSI implementation needs at least 7 devices, 4 for the counter and holding register, 2 for flipflops, and 1 for mux. Mounting this many devices, their associated decoupling capacitors, and PCB real estate is costly. The alternative is a low cost CPLD. We need 4 macrocells for the synchronizer chain and about 10 for bus interface, leaving us either 18 or 52 macrocells for the counter and holding register in the two smallest Xilinx Coolrunner parts. A 9 bit signal edge counter may suffice if a sufficiently large prescaler is used. With an arbitrarily chosen 20kHz limit on the lenghtening interrupt the upper limit becomes 2^9*20kHz=10.24MHz. The cost is ~e1 and ~e3 for the two parts.

Interpolator

Designing an interpolator without exotic components limits selection severely. The Coolrunner CPLD has relatively slow but constant, risetimes, and we need a switch that has picosecond-range switching uncertainty. The 1N4148 diode is such a part. BAS16 is similar. They are 200mA 100V types marketed as high speed switching diodes, and likely the most common diode around. While the datasheet states 4ns switching speed, the switching uncertainty is much less. By using three of these diodes in combination with the CPLD, we can create a simple current-gated integrator circuit that has less than 50ps noise:

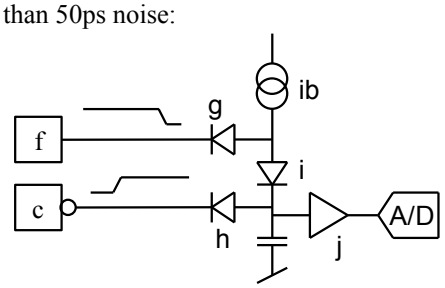


fig. 2

When idle, the output of */c* is low, and diodes *h* and *i* are conducting. The capacitor sees a low impedance, consisting of the */c* output impedance and the impedance of diode *h* in series, and is preset to the starting value. When the gate starts closing */c* goes high. Diode *h* turns off, and the capacitor starts charging. When *f* goes low diode *g* steals the current from the current source, diode *i* turns off, and the charging stops. Since diodes *h* and *i* operate into a capacitive load, and at the same current, the reverse recovery charges mostly cancel: When *h* turns off, additional charge is injected into the capacitor, then *i* turns off and steals a similar charge. Likewise the voltage dependant capacitance of diodes *h* and *i* are opposite with respect to integrator voltage, mostly canceling nonlinearity from varactor effects.

The charge left on the capacitor must be quantitized before it leaks away through parasitics and diode leakage. With a 1nF capacitor and a leakage of 100 nanoamperes the resulting slew rate is 100V/s, so for a 1mV error the amplifier *j* and the A/D converter needs to settle and do sample/hold in less than 10 microseconds. With a 200kHz A/D clock the AVR series can do this, though it should be noted that as the leakage current in the diodes is reltively constant, sampling at twice that does not degrade linearity significantly.

With a 8mA current source and 1nF capacitor, the change in capacitor voltage is 8mV/ns, yielding 320 ps resolution if applied to the A/D converter directly. The test circuit used a TLC072 as straight buffer, and the output when the counter was fed with its own reference clock was flicker free. Thus the peak to peak noise of the synchronizer chain, Coolrunner CPLD input, output, and diodes, does not exceed ~50 ps RMS.

When applying signals other than the reference clock, the figure deteoriates to ~100ps since the beat between noise generated by the reference, and noise generated by the signal applied, causes shifts in input thresholds, and thus in arrival timing.

Calibrating the interpolator

The system needs to be self calibrating since a separate calibration process would add cost. By running the counter against its own reference, and using the bypass and non-bypass states of multiplexer *e*, the cpu can acquire the capacitor voltages resulting from two arrival times exactly one reference clock cycle apart. When the A/D converter value is divided by this difference, and subtracted from the value in the reference holding register (in the test circuit the TCNT1 counter in the AVR cpu), the result is correct scaling.

The temperature coefficient of diode *h* and the nmos transistor in the CPLD output */c* also cause drift, in the test circuit this proved to be on the order of 1ps/s. This can be compensated out by occasionally doing a second A/D conversion after the circuit has been reset to the starting state, thereby negating the drift without the penalty of halving the acquisition rate for signals faster than half the A/D conversion rate.

Linear regression

Since a frequency counter is supposed to measure the ratio between two frequencies, and not the time it takes for some number of cycles to pass, performance of the counter can be improved by sampling many times over the duration of the acquisition period, and not just at the beginning and end. Since the AVR A/D can do around 15k conversions per second, and the effective noise drops with roughly the square root of of the number of acquistions, with a one second gate time the equivalent flicker free start-stop uncertainty drops from ~320ps to ~2.6ps. Thus, using linear regression, the test circuit provided 12 flicker free digits at 1 second gate time.
If linear regression is applied to a counter without interpolator, the beat between the applied signal and the reference produces input dependant flicker, and input dependant reduced precision. This is most pronounced when the ratio between the signal and the reference is (almost) an integer.

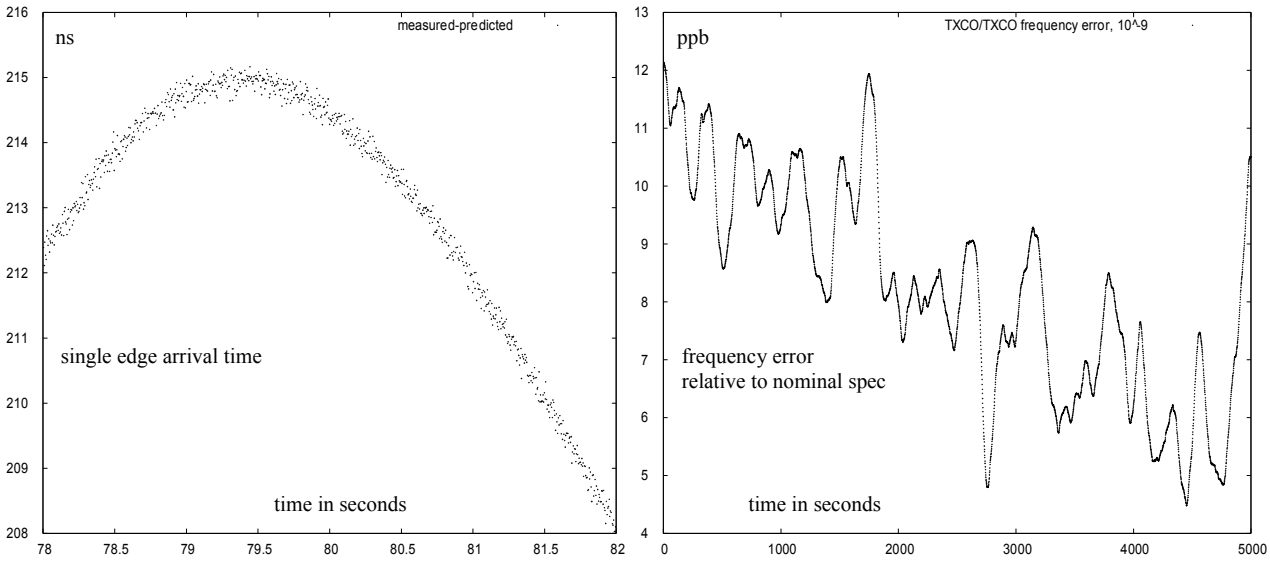
Where the reciprocal counter needs 100 seconds to do a 1ppb measurement, the similar reciprocal interpolating regressing counter provides 1000 times better precision in 1/100th the time.

Implementations for slow input signals

If the input signal bandwidth is low, low defined as less than the conversion rate of the interpolator and A/D converter, the signal counter and signal holding register can be eliminated, leaving just the three synchronizer flipflops *c,d,f* and multiplexer *e*. Keeping multiplexer *e* allows self calibration, thus for applications like PPS inputs in GPS disciplined oscillators the added cost of providing sub-nanosecond quantization is set by the four flipflops, the capacitor, the current source, and the buffer. The cost saving going from a sub-nanosecond input to a poor 100ns one is ~e1.5. Compared to the cost of other components in a frequency counter system, an interpolator provides a lot of performance per euro.

Closing notes

Neither the interpolating reciprocal counter nor the regressing counter are new. They are, however, regarded by most engineers as somewhat of a black art, the domain of custom silicon and expensive laboratory instruments. In reality interpolating counters and capture systems can be built with cheap components already in stock, at low cost, while still yielding good performance. Why settle for less?



Left: Timestamping a 18.414MHz TCXO against a 10MHz DCF-DO: ~500 ps p-p, mostly due to A/D quantization.
Right: Comparing a 10MHz TCXO and a 18.414MHz TCXO with 1 second gate time.

VHDL source for test implementation

```
----------------------------------------------------------------------------------
--
-- Company:
-- Engineer:       Kasper K. Pedersen
--
-- Create Date:    23:32:39 01/04/2008
-- Design Name:    Timestamper
-- Module Name:    top - Behavioral
-- Project Name:   Targeting LIDAR
-- Target Devices: 64MC Coolrunner, 64MC CoolrunnerII
-- Tool versions:  9.1i.J.30
-- Description:    Event timestamper with two non-simultaneous channels
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
    Port ( refclk    : in  STD_LOGIC;  --
           cputx     : in  STD_LOGIC;  -- not related to T/F
           cpurx     : out  STD_LOGIC; -- not related to T/F
           cputen    : in  STD_LOGIC;  -- not related to T/F
           transrx   : in  STD_LOGIC;  -- not related to T/F
           transtx   : out  STD_LOGIC; -- not related to T/F
           transen   : out  STD_LOGIC; -- not related to T/F
           miso      : out  STD_LOGIC; -- SPI shift out of hold reg
           mosi      : in  STD_LOGIC;  -- SPI shift-in of configword
           sck       : in  STD_LOGIC;  -- SPI shift clock
           ss        : in  STD_LOGIC;  -- 0 resets, 1 to captures
           icp1      : out STD_LOGIC;  -- capture strobe to ref ctr/status
           pd4       : in  STD_LOGIC;  -- nc
           pd5       : in  STD_LOGIC;  -- nc
           int1      : out STD_LOGIC;  -- signal ctr elongation
           pd7       : in  STD_LOGIC;  -- nc
           tosc2     : in  STD_LOGIC;  -- nc
           modeshift : in  STD_LOGIC;  -- 1 to load config word
           led1      : out  STD_LOGIC; -- capture indicator
           led2      : out  STD_LOGIC; -- not related to T/F

           rampstopN : out STD_LOGIC;  -- interpolator outputs
           rampstartP: out STD_LOGIC;

           sigclk    : in STD_LOGIC;   -- input to signal counter, connected
           sigout    : out STD_LOGIC;  -- to this pin, the input mux output
           sigin1    : in STD_LOGIC;   -- input 1
           sigin2    : in STD_LOGIC);  -- input 2
end top;

architecture Behavioral of top is
  signal mux: STD_LOGIC_VECTOR(3 downto 0);
  signal eventcounter,eventhold: STD_LOGIC_VECTOR(15 downto 0);
  signal readindex:STD_LOGIC_VECTOR(3 downto 0);
  signal syn1siggate,siggate:STD_LOGIC;
  signal syn1cpugate,cpugate:STD_LOGIC;
begin
```

```
--mux: input source 1, 2, REF, or off
multiplexer: process(sigin1, sigin2, mux)
begin
  if mux(1 downto 0)="00" then
    sigout<=sigin1;
  elsif mux(1 downto 0)="01" then
    sigout<=sigin2;
  elsif mux(1 downto 0)="10" then
    sigout<=refclk;
  else
    sigout<='0';
  end if;
end process;

--ct: count edge number on source
signalcounter: process(sigclk)
begin
  if RISING_EDGE(sigclk) then
    eventcounter <= eventcounter +1;
    if siggate='1' then
      eventhold<=eventcounter;
    end if;
  end if;
end process;

--mux for serial readout of source edge hold register to cpu
readselect: process(readindex)
begin
  case readindex is
    when "0000" =>  miso <= eventhold(0);
    when "0001" =>  miso <= eventhold(1);
    when "0010" =>  miso <= eventhold(2);
    when "0011" =>  miso <= eventhold(3);
    when "0100" =>  miso <= eventhold(4);
    when "0101" =>  miso <= eventhold(5);
    when "0110" =>  miso <= eventhold(6);
    when "0111" =>  miso <= eventhold(7);
    when "1000" =>  miso <= eventhold(8);
    when "1001" =>  miso <= eventhold(9);
    when "1010" =>  miso <= eventhold(10);
    when "1011" =>  miso <= eventhold(11);
    when "1100" =>  miso <= eventhold(12);
    when "1101" =>  miso <= eventhold(13);
    when "1110" =>  miso <= eventhold(14);
    when "1111" =>  miso <= eventhold(15);
    when others =>  miso <='0';
  end case;
end process;

--SPI interface towards the cpu, counter controlling the mux above,
--and shift-in of the 4 bit configuration word controlling
--the source multiplexer and the two muxes in the sync chain
readclock: process(sck,readindex)
begin
  if RISING_EDGE(sck) then
    readindex <= readindex + 1;
    if modeshift='1' then
      mux <= mux(2 downto 0) & mosi;
    end if;
  end if;
  if ss='0' then --async clear on 0
    readindex<="1111";
  end if;
end process;
```

```
  gatesig: process(sigclk, ss)
  begin
    if RISING_EDGE(sigclk) then
      syn1siggate <= not ss;        -- ss 9 is prepare,1 is do-capture
      siggate <= syn1siggate;       -- 2FF-synchronizer mode
      --gate is signal-synchronous, goes low now, locking the counter.
      if mux(3)='1' then
        siggate <= not ss;          -- 1FF-synchronizer mode
      end if;
    end if;
    --async reset: gate ON when ss goes low.
    if ss='0' then
      syn1siggate <= '1';
      siggate <= '1';
    end if;
  end process;

  gatecpu: process(refclk)
  begin
    if RISING_EDGE(refclk) then
      syn1cpugate <= siggate;
      cpugate <= syn1cpugate;     --cpugate - siggate = 1..2 clock
      if mux(2)='1' then
        cpugate <= siggate;       --0..1 clock (for interpolator calibration)
      end if;
    end if;
    --no async when ss is 0, no need, after 2 clk it has propagated.
  end process;

  rampstartP <= not siggate;      --interpolator outputs
  rampstopN <= cpugate;

  icp1 <= cpugate;                --capture indicator
  int1 <= eventcounter(15);       --signal ctr elongation.

  transtx <= cputx;
  cpurx <= transrx or not cputen; --cputen low=don't rcv.
  transen <= not cputen;          --inverted! low is transmit
  led2 <= not ss;                 --show capture status on LED
  led1 <= cputen;                 --activity indicator
end Behavioral;
```

52 MC, 76 PT, 45 FF, 87MHz sigclock max, 95MHz refclock max in the cheapest part.