An generalized implementation of the squarewave filtering algorithm f

```
/*
 Correlate an analog signal to a squarewave, returns the amplitude of the match

 Inputs: pin - Analog pin to read from
 FREQ_PERIOD - the period in us of the desired frequency to filter
 NUM_READINGS - The number of samples to read
 SAMPLE_INTERVAL - the amount of time in between each sample in microseconds
 NUM_OFFSETS - the total number of parallel, offset dot products to be performed
 *Note read rates below 5-6 are not possible with analogRead()*/

double takeSquareSignalSample(byte pin, int FREQ_PERIOD, int NUM_READINGS,
                              int SAMPLE_INTERVAL, int NUM_OFFSETS)
{
  // Record start time of a cycle and the first sample begin time
  static int32_t tStart = micros(); // us; start time of the last analog read
  static int32_t tbegin = tStart;   // The begining time of the reference signal

  double readVal = 0;     // Value read from pin
  int i = 0;              // index of reading
  int cyclePosition = 0; // position of sample in a single wave period

  double dotProd[NUM_OFFSETS];    // accumulators to store dot product
  int sampleMult[NUM_OFFSETS];    // The value of the reference signal for each offset
  int phaseOffsets[NUM_OFFSETS]; // The amount of phase offset from 0 for each offset


  // Calculate the phase offset values to spread the total evenly
  // within the first half of sample period of the desired frequency
  for (int j = 0; j < NUM_OFFSETS; j++)
  {
    // Initialize all dot product accumulators to 0
    dotProd[j] = 0;
    // Evenly space the phase offsets over the first half of the signal
    phaseOffsets[j] = (FREQ_PERIOD * j) / (2 * NUM_OFFSETS);
  }
  // For 4 offsets for example, the phase offset and multipliers are
  // as follows for a 100us period signal (10kHz):
  /*
  // 1st sample multiplier (correlation to expected 0  phase signal) -> 0
  // 2nd sample multiplier (correlation to expected 12 phase offset signal)
  // 3rd sample multiplier (correlation to expected 25 phase offset signal)
  // 4th sample multiplier (correlation to expected 37 phase offset signal)
  */



  // Loop to perform concurrent sampling and correlation / dot product
  // This is the repeated part of the sequence that samples at fixed intervals while
  // computing the dot product of parallel streams
  while (i < NUM_READINGS)
  {
    // Only take a reading once per sample interval
    int32_t tNow = micros(); // us; time now
    if (tNow - tStart >= SAMPLE_INTERVAL)
    {
      // reset start time to take next sample at exactly the correct spacing
      tStart += SAMPLE_INTERVAL;
```

```
58
59        readVal = analogRead(pin);
60
61        // Account for the shift on the wave signal so far
62        // Take mod to normalize back into the domain of a single wave period
63        cyclePosition = (tNow - tbegin) % FREQ_PERIOD;
64
65        // Update the correlation to the reference signal
66        // based on the cycle position for all phase offsets
67        updateReferences(sampleMult, phaseOffsets, FREQ_PERIOD, NUM_OFFSETS, cyclePosition);
68
69        // Perform the next step of the concurrent dot products (one for each offset amount)
70        for (int j = 0; j < NUM_OFFSETS; j++)
71        {
72          dotProd[j] += readVal * sampleMult[j];
73        }
74      }
75      // Index is complete, move to next one
76      i++;
77    }
78
79    // Combine the dot products using the root mean squared
80    double rms = 0;
81    for (int j = 0; j < NUM_OFFSETS; j++)
82    {
83      rms += dotProd[j] * dotProd[j];
84    }
85      rms = sqrt(rms);
86
87    // normalize and return value
88    return rms * 2.0 / (NUM_READINGS);
89  }
90
91
92  // This is the square correlation, if the sample including offset
93  // falls within the first half of a wave period, then it is positive, otherwise negative
94  void updateReferences(int sampleMult[], int phaseOffsets[], int FREQ_PERIOD,
95                        int NUM_OFFSETS, int cyclePosition)
96  {
97    for (int i = 0; i < NUM_OFFSETS; i++)
98    {
99      if (((cyclePosition + phaseOffsets[i]) % FREQ_PERIOD) < (FREQ_PERIOD / 2))
100     {
101       sampleMult[i] = 1;
102     }
103     else
104     {
105       sampleMult[i] = -1;
106     }
107   }
108 }
```