

MARK 5 MEMO #021

**MASSACHUSETTS INSTITUTE OF TECHNOLOGY**  
**HAYSTACK OBSERVATORY**  
WESTFORD, MASSACHUSETTS 01886  
November 30, 2004

*Telephone: 978-692-4764*  
*Fax: 781-981-0590*

To: Mark 5 Development Group

From: Roger Cappallo

Subject: Phase Cal Extraction for the Mark 5B

**Table-driven method (AEER's Legacy)**

Since the Mk5B will replace the Station Unit (SU) subsystems on the Mk4 correlator, it is necessary to provide a phase cal tone extraction capability. One approach would be to implement the scheme that was used in the VLBA pcal extractor, and also within the SU's, as originally suggested by Alan Rogers (Mk4 memo #176). He uses a table in RAM to hold the products of all possible input samples with the desired pcal tone waveform, and for each sample adds in the appropriate product into accumulators. In order to conserve RAM, the lookup table is only 3200 entries long, which restricts the allowable pcal tone frequencies to be multiples of 10 KHz (thereby repeating every 100 us, which is 3200 samples of the 32 MHz sample clock). Another compromise within the method is the representation of the products as (only) 2-bit quantities, which causes about 2.1% of the power to be aliased to different tone frequencies.

In the SU pcal module the implementation of Alan's method required 8 Xilinx XC-4010 chips, each of which have 400 CLB's. The FPGA used in the Mk5B is the XC2VP30, which has 3680 CLB's; therefore, it is reasonable to think that the design would fit into a single chip, but with few resources to spare. Such a device would extract two tones from each of the 16 channels.

**Mixer-based method**

An attractive alternative to the above design is to make use of the high speed multipliers on the VP30. Ideally, we would like to be able to extract as many as 16 tones from each channel, since the tones are normally at 1 MHz spacing, and the maximum Mk4 video bandwidth is 16 MHz, yielding 16 in-band tones. By using the 18x18 multipliers we can not only extract all 256 of these tones, but do so with high fidelity – using a 64-level waveform to represent a sinusoid.

In order to conserve multiplier resources I've used a small trick to do two complex multiplications in each product, and I'm also proposing to time-multiplex the use of the multipliers, running them at 128 MHz to process four 32 MHz streams. If two 2-bit input samples from independent frequency channels are a and b, I suggest packing them in an

18 bit word as (aa00000000000000bb). Furthermore, if the real and imaginary components of the phasor are c and d, they are packed as (0000cccccc00dddddd). Then the full 18 bit product yields:

$$(a \times 2^{16} + b)(c \times 2^8 + d) = ac \times 2^{24} + ad \times 2^{16} + bc \times 2^8 + bd$$

The four 8 bit products can be simply plucked out of the result. Using this scheme, as well as the x4 time-multiplexing, requires only 32 multipliers (out of 136 on a VP30). If instead we decide to run the multipliers at a more leisurely 64 MHz, then this approach consumes 64 multipliers.

Since we are using unsigned numbers to represent biased, signed data values, we must make a correction to the dumped values, after multiplying and summing. Let  $x_i$  and  $y_i$  represent two data values, which are biased by  $\alpha$  and  $\beta$ , respectively. (For the case of 2 bit data samples, the bias  $\alpha$  is 1.5). Consider the relationship

$$\sum_i^N (x_i + \alpha)(y_i + \beta) = \sum_i x_i y_i + \beta \sum_i x_i + \alpha \sum_i y_i + N\alpha\beta$$

The first sum on the right hand side is that which we desire. Since we are keeping track of the number of samples in each state, we can predict the values of the other terms on the RHS, and by subtracting them from the actual sum on the LHS, which is returned, we can derive the quantity of interest.

*Note: There is a potential problem here. The Virtex II FPGA's apparently use 2's complement arithmetic in their 18x18 bit multipliers. If the high order bit of a is 1, representing a sample with a negative voltage value, the unsigned multiplication table is corrupted by the ensuing sign extension. More study is needed.*

## **Tone Generation**

The task of the tone generator is to generate, in flexible fashion, quadrature components of an arbitrary tone. It does so (see Figure 3) by maintaining a phase register, whose content is used as the independent variable in a table lookup of the complex phasor. By choosing a phasor with 64 levels peak-to-peak, and quantizing the phase into 64 steps per quarter cycle (see Figure 4) the errors can be kept quite small: it was found via Matlab simulation that the resultant correlation error is  $1.1 \times 10^{-4}$  in voltage-like units. Thus the total of all spurious tones generated at harmonic frequencies is down from the fundamental by 40 dB.

The "rotator" is much like the Mark 4 correlator rotator, except that it is only a linear model since there is no acceleration of the tones. It consists of a 32-bit phase register, to which is added a phase increment register on each clock cycle. The phase register holds phase in units of  $2^{-32}$  rotations, so there is an implied decimal point at the left-hand edge of the register (in a big-Endian world). It is not necessary to be able to set the initial value of the phase register, as any necessary complex rotations (such as might be needed for non-integral tone frequencies) can be applied in software. It is sufficient to reset the phase register to 0 on each second tick.

## Quantization Error Analysis

Quantization errors can arise from several sources in the mixer-based scheme:

1. level quantization of the sine-wave
2. phase quantization of argument in sine-wave lookup
3. frequency quantization of sine-wave

The result of any of the above quantization errors will be to introduce sensitivity to signals at frequencies other than the intended phase cal tone frequency. This arises through the impurity of the synthesized sine wave, having harmonics at other, well-defined frequencies. Numerical evaluations of the correlation error in voltage-like units have been made in Matlab, for the first two (level and phase) effects listed above; the results can be seen in Table 1. Based upon this analysis we've chosen to use 64 sine wave levels peak-to-peak, and 64 phases within a quarter cycle of the sine wave.

phases levels	16	32	64	$\infty$
32	841	449	362	344
64	458	177	109	83
128	419	119	45	20

**Table 1** Correlation error in units of  $10^{-6}$ , as a function of the number of discrete phases and levels employed. Column headers are the number of phase points per quarter cycle; row headers are discrete level steps within the interval  $[-1, 1]$ .

Updating the rotator phase on every sample clock will result in fringe rate quantization at the level of  $32 \times 10^6 / 2^{32} = 7.451$  mHz; thus the maximum error is  $\pm 3.725$  mHz. The error due to such a phase-rate granularity was found numerically within Maple, by evaluating the analytic expression

$$corr = \frac{\int_0^1 (\sin(\omega t) \sin((\omega + \delta\omega)t - \delta\omega/2)) dt}{\left[ \int_0^1 (\sin(\omega t))^2 dt \cdot \int_0^1 (\sin((\omega + \delta\omega)t - \delta\omega/2))^2 dt \right]^{1/2}}$$

for various values of  $\omega$  and  $\delta\omega$ . Note that the mean phase offset of  $\delta\omega/2$  doesn't actually need to be applied in the hardware, as the complex results can be rotated in post-processing software. The resulting errors for the 10 KHz tone (i.e.  $\omega = 2\pi \times 10000$ ), which was  $23 \times 10^{-6}$  for the maximum possible fringe rate quantization error of 3.725 mHz. The error levels for other integral tone frequencies were similar to those for the 10 KHz tone. In comparison to the level and phase quantization errors in Table 1, this effect is seen to be negligible.

## 8x1 Correlation Cell

The 8x1 correlation cell shown in Figure 5 is the central engine of the tone extraction correlator. It processes the data for eight channels of 32 MHz data, by multiplying two at a time in a time-multiplexed 128 MHz circuit. The same complex phasor value from the tone-generator is applied to each of the four pairs of data points.

## State Counts

The number of occurrences of each of the four sample states is needed in order to do proper normalization of correlation results. As can be seen in Figure 6, there are 64 counters dedicated to totaling the number of samples in each of 4 states for each of the 16 input channels. Whenever a valid sample is decoded by the “decode SMV” block, the appropriate output line is raised, and the corresponding counter is incremented. Since the counters run at 32 MHz for up to 1 second between dumps, 25 bits is both necessary and sufficient for the counter size. It is possible that a single adder may be used per channel, if the correct operand can be fetched, incremented and stored in 31.25 ns.

## Resource Usage

The following FPGA resources are used for the pcal and state count circuits:

- 32 multipliers (hardware 18 bit multipliers; clocked at 128 MHz)
- 128 adders (8 bits added into 32bits; clocked at 128 MHz)
- 512 accumulators for 32 bit quadrature components (16 Kb RAM total)
- 768 bits of RAM for sinusoidal waveform table (64 phases x 12 bit phasors)
- 16 x 32-bit full-adders (32 MHz clock)
- 16 x 2 x 32-bit phase and phase\_inc registers (1024 bits RAM)
- 64 (or possibly 16) x 25 bit counters @ 32 MHz

## I/O to phase cal subsystem

### Input signals

- 16 channels of 2-bit (SM) data @ 32 MHz clock rate
- 32 MHz clock
- global validity with single sample resolution
- 1 PPS active on start of each TOT second

### Input parameters

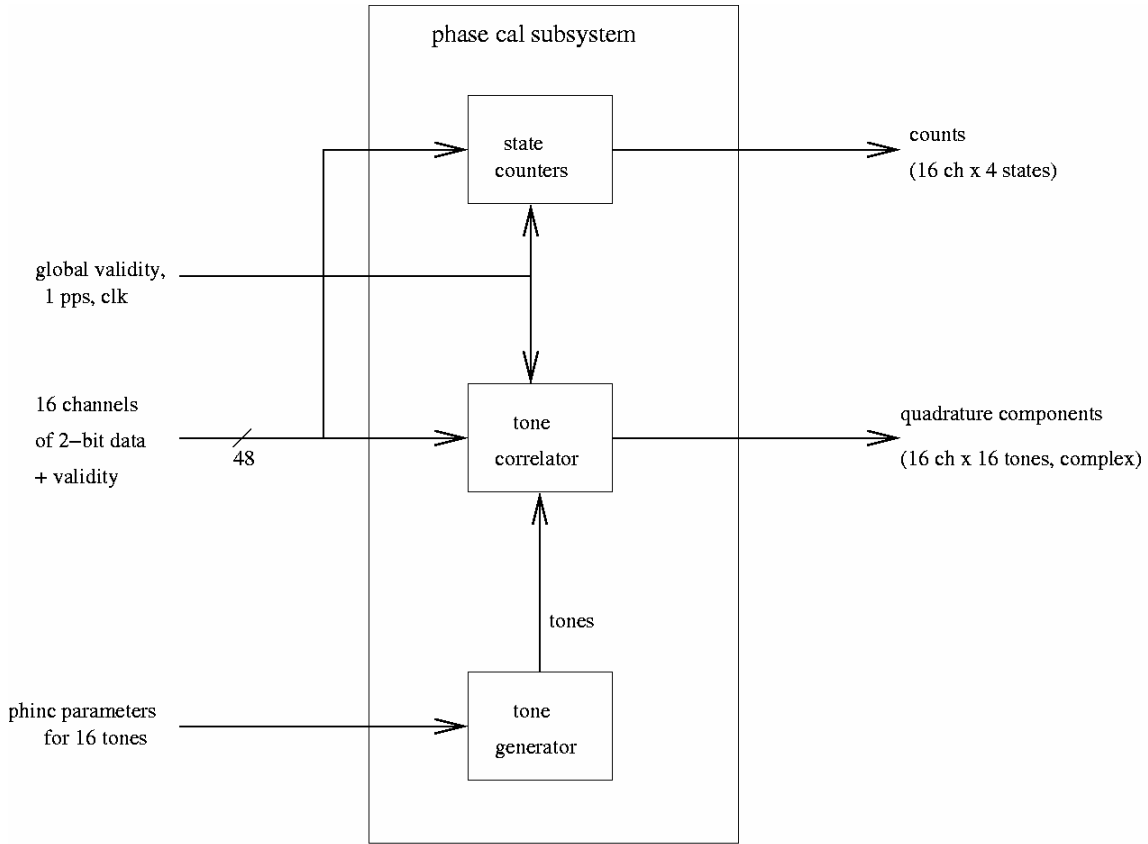
- phase increment – 32 bits, updated @ 1 Hz

### Output signals

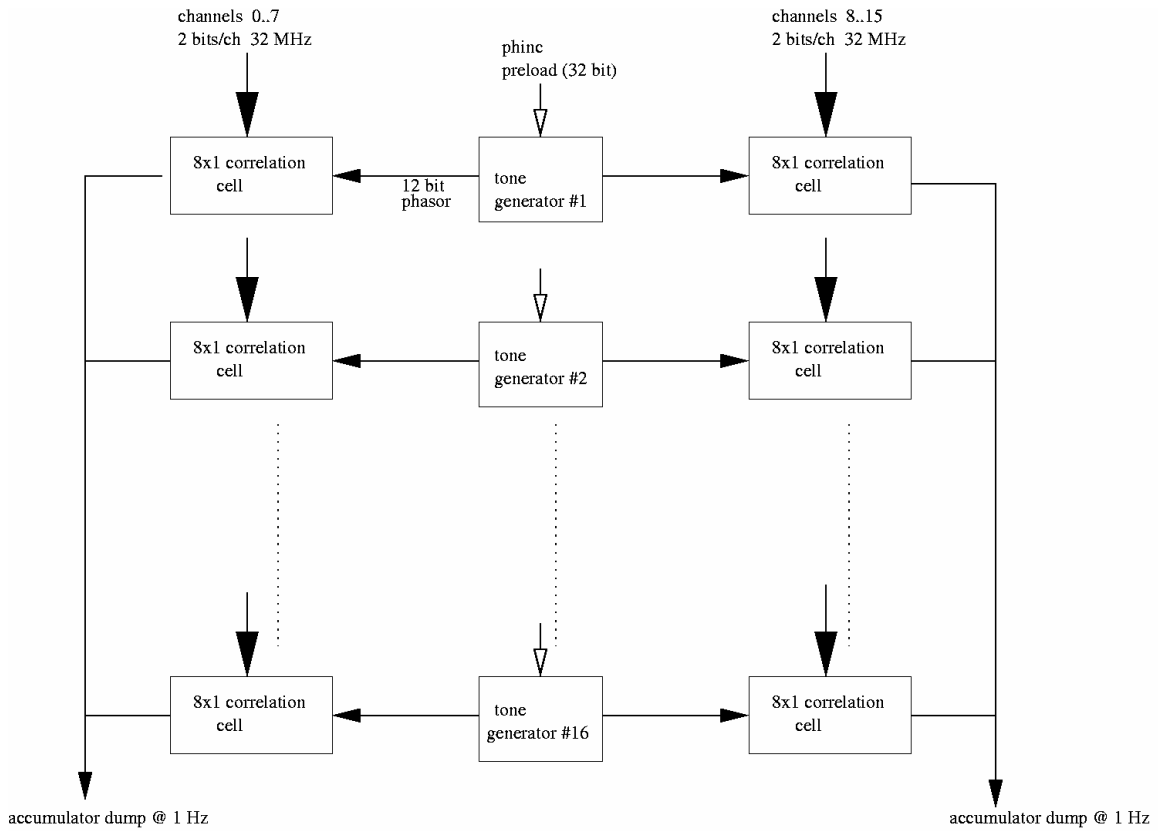
- *none*

## Output data

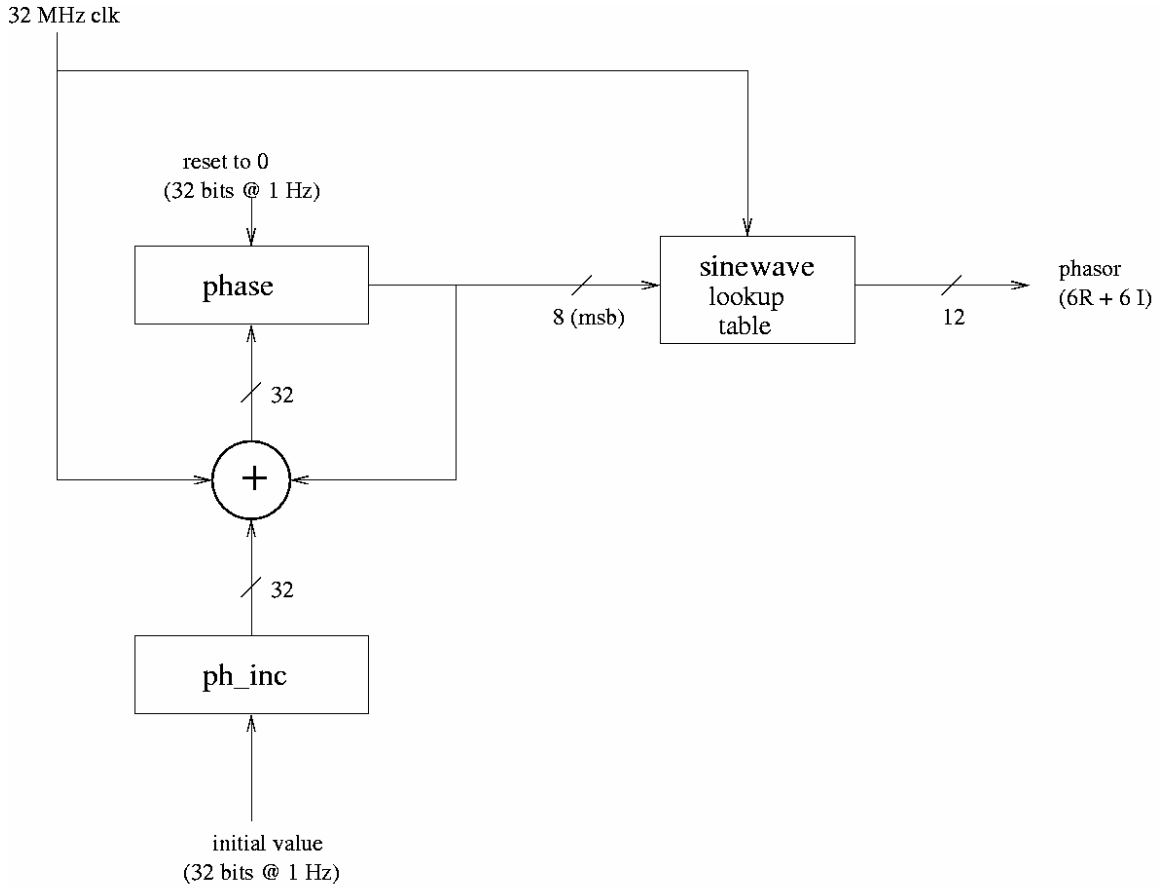
- real & imaginary 32-bit counts for 16 tones x 16 channels (512 counters total), dumped @ 1 Hz
- state counts: 16 channels x 4 states, 32 bits, dumped @ 1 Hz



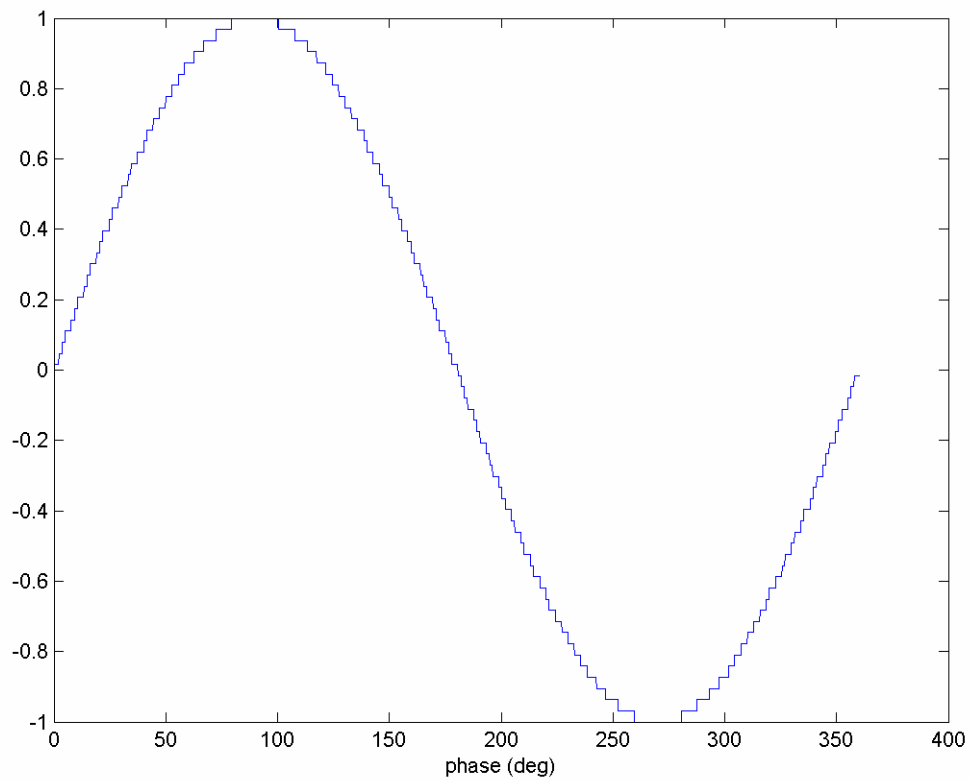
**Figure 1 Top level view** of the phase cal subsystem, showing its external interface to the rest of the Mk5B. The 16 phase increment parameters, 512 quadrature components, and 64 state counts get updated once/s.



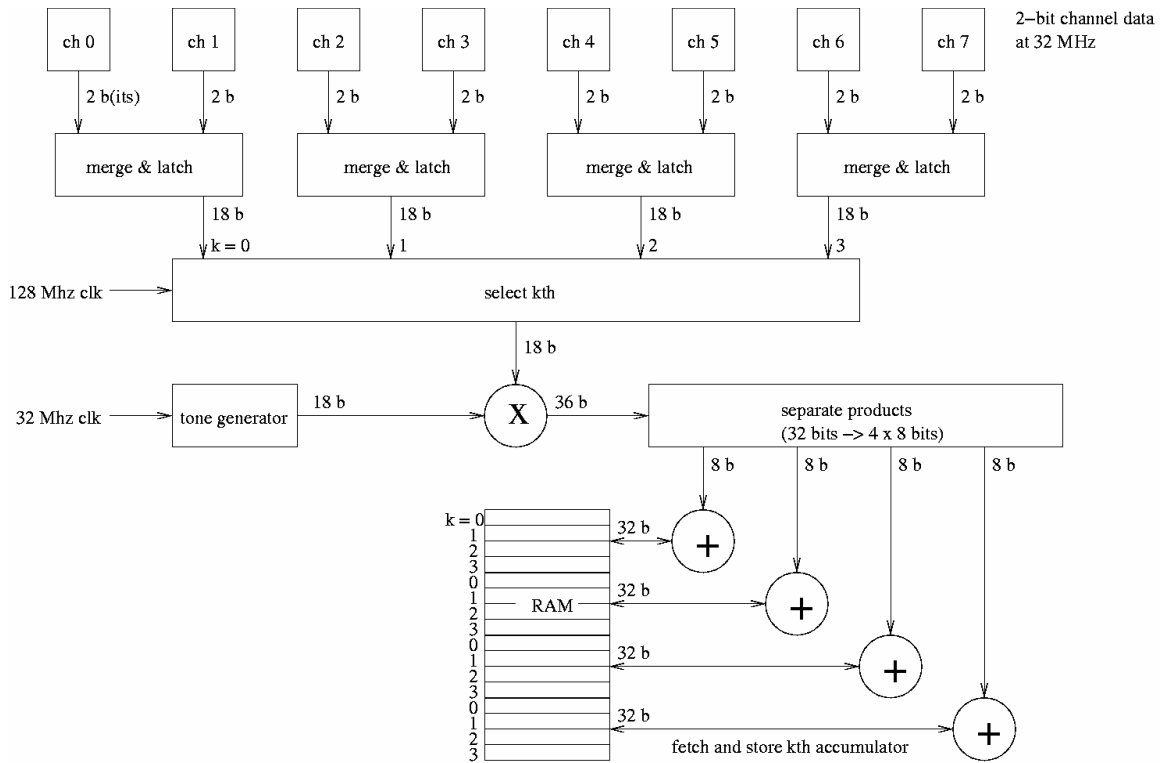
**Figure 2 Tone generators and correlators** – The same set of 8 two bit channel signals is applied to each correlation cell, while each tone generator receives an independent phase increment once/s. In this manner each data stream can have 16 different tones extracted from it.



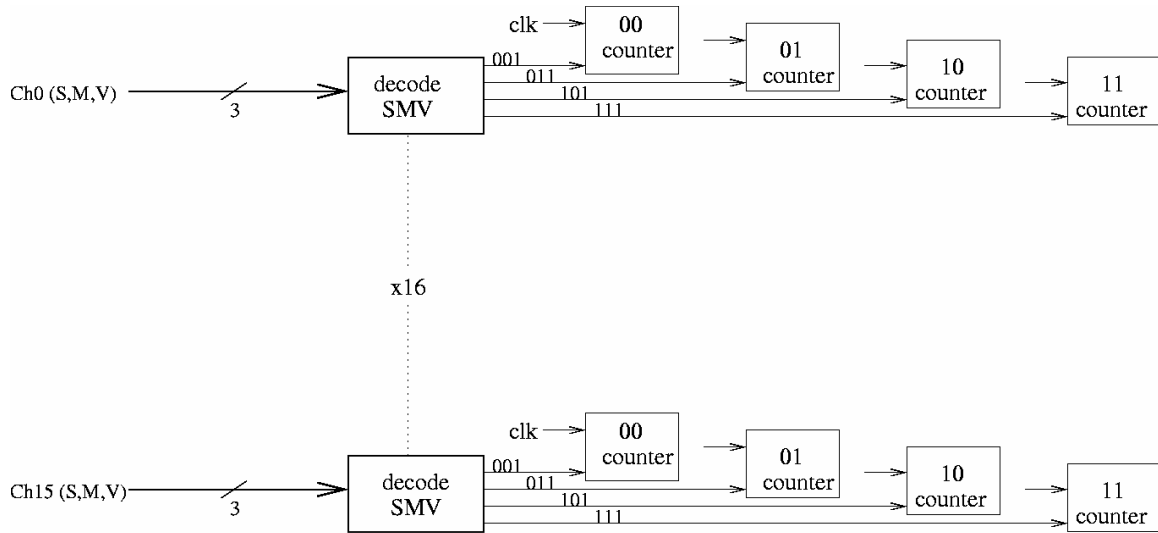
**Figure 3 Tone Generator** – The tone generator is based upon a numerically controlled oscillator, in which a phase increment is added upon every 32 MHz clock into a phase register. The 8 most significant bits (which gives a phase resolution of 1/256 of a cycle) are then used to look up the components of a sinewave phasor from a table in RAM. The phasor has 6 real and 6 imaginary bits, possibly already stored as a 16 bit field, with 0-padding, in order to facilitate a simpler implementation.



**Figure 4 Sinusoidal waveform** having 64 quantized levels and whose phase is quantized using 8 bits / cycle (~1.4 deg). Both levels and phase are rounded to the nearest digital value. Spurious tones are suppressed by > 40 dB.



**Figure 5 8x1 Correlation Cell** – Within the correlation cell for one tone, eight different channels of data are extracted. Since there are a total of 16 channels of data, as well as 16 tones, the system has a total of 32 such 8x1 correlation cells. Within each cell, the channels are correlated two at a time by utilizing the wide word width in the multipliers, and are time-multiplexed by an additional factor of four. Thus 4 correlations are done in each 32 MHz clock cycle. In this drawing, the multiplier and all 4 adders are running at 128 MHz, with the index  $k$  determining which of the 4 multiplexed operations is going on during each 32 MHz clock cycle. The product should only be added in for valid samples.



**Figure 6 State Counters** – Each of the 16 channel data streams is analyzed for the number of occurrences of each data state. This can be done simply by incrementing the appropriate counter whenever the corresponding data pattern has been sensed, for each valid sample. The 64 output counts are dumped each second.