PEP-II BPM

1. Introduction

2. BPM processing

3. Requirements and functionality
   - 3.1. Resolution
     - 3.1.1. Thermal noise
     - 3.1.2. Electronics
     - 3.1.3. Electronics reflection
   - 3.2. Functionality

4. Filter-Isolator Box (FIB)

5. Processing module (Ring I&Q - RInQ)
   - a. RF I&Q demodulator
     - i. Requirements and functionalities
     - ii. Block Diagram and circuit characteristics
   - b. Calibrator
     - i. Requirements and functionalities
     - ii. I&Q processing errors
       1. Amplitude unbalance
       2. Phase unbalance
     - iii. Block Diagram and circuit characteristics
   - c. Local oscillator
   - d. Baseband processor
     - i. DSP Program and Data Memory (32k x 32)
     - ii. Boot Memory (32k x 16)
     - iii. Dual Port Memory (16k x 16)
     - iv. ADC Interface
     - v. DSP
     - vi. CAMAC Functions
     - vii. Memory Map

6. Software algorithm
   - e. Remote mode
     - i. Set the channel
     - ii. Set the local oscillator
       1. Set 16-bit mode
       2. Set frequency
       3. Set phase
     - iii. Set calibrator
       1. Set 16-bit mode
       2. Set calibrator ON

1. Introduction

The Beam-Position Monitor (BPM) system for the PEP-II B-Factory at SLAC must measure closed orbit positions for a 3-A multibunch beam and turn-by-turn positions for a low current single bunch injected in a 200-ns gap in the multibunch beam. A system that combines broadband and narrowband capabilities and provides data at high bandwidth was designed. The electronics must also have a very low reflection coefficient, in order to reduce interference from multibunch when the single bunch measurement is required. The system includes the Filter-Isolator Box (FIB) that selects a harmonic of the bunch spacing (952 MHz) and absorbs the other frequency components and the Ring I&Q (RInQ) module. The RInQ is a CAMAC based wideband I&Q demodulator, ADC, and signal processing unit that provides beam position information to the Control System. A calibrator that must work in the presence of beam, correcting for electronic measurement errors, measurement offset, and gain is incorporated on board. This paper describes the system requirements, the electronics design, and the laboratory tests.

2. BPM Processing

A block diagram of the BPM system is shown in Figure 2.1 (y-only BPMs are shown). There are three styles of BPMs x-only, y-only, and xy. Each RInQ module has 8 inputs and can serve 2 xy BPMs or 4 x-only (y-only) BPMs.

The cables are color coded and are connected in the same way at the electrodes, while they do not have the same connection at the FIBs. The transition panel and the RInQ processor keep the same layout (top connector to top connector and so on; except for xy BPMs where blue and green are swapped). The RInQ front panel has LER on the top and HER on the bottom, plus sign on the top and minus sign on the bottom.

In Figure 2.2 the system layout for x-only and y-only BPMs is shown and in Figure 2.3. the system layout for xy BPMs is shown. Note that for both HER and LER the positive x direction points outward from the ring.

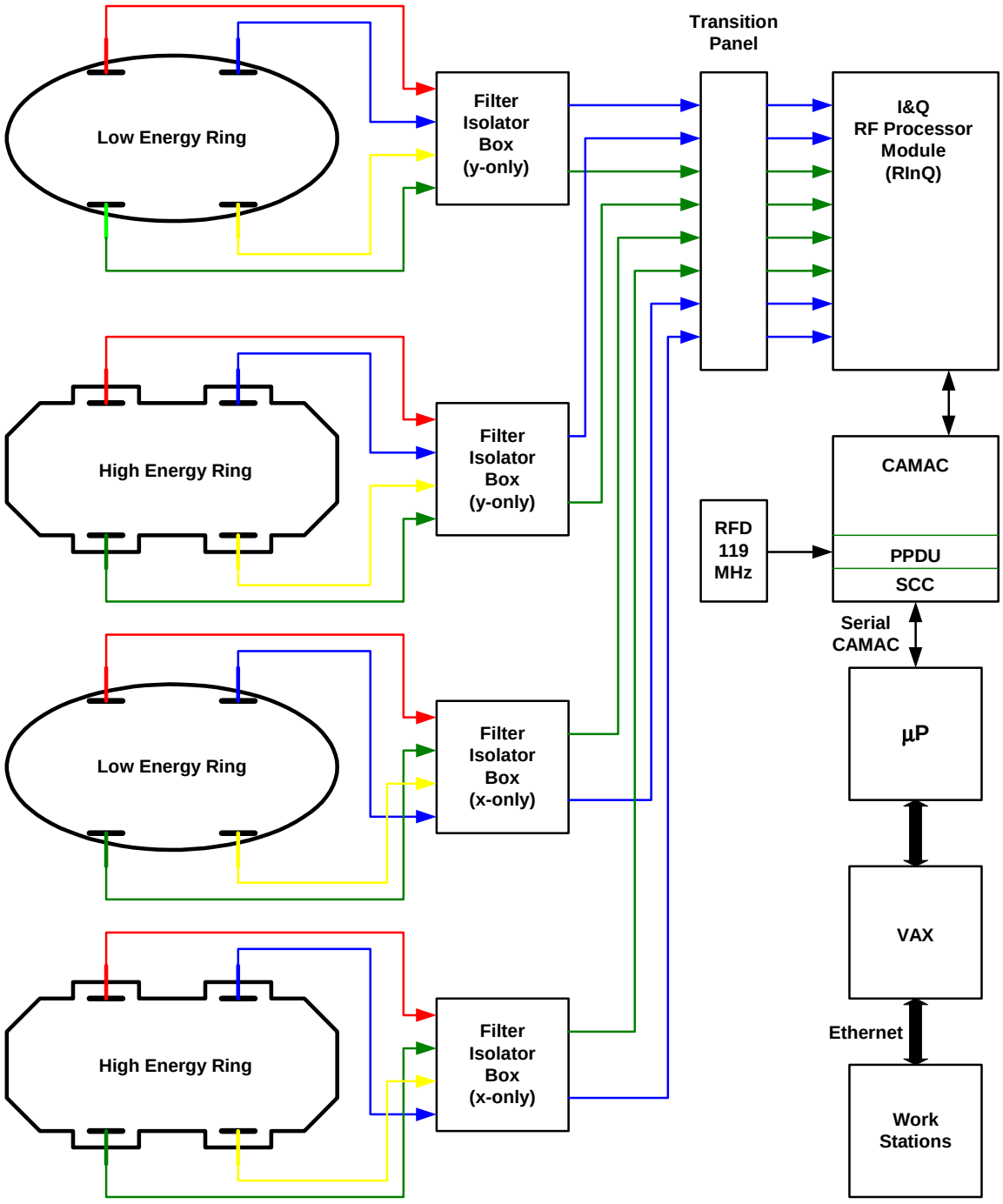

Figure 2.1 BPM system block diagram (Top: y-only BPMs; Bottom: x-only BPMs).

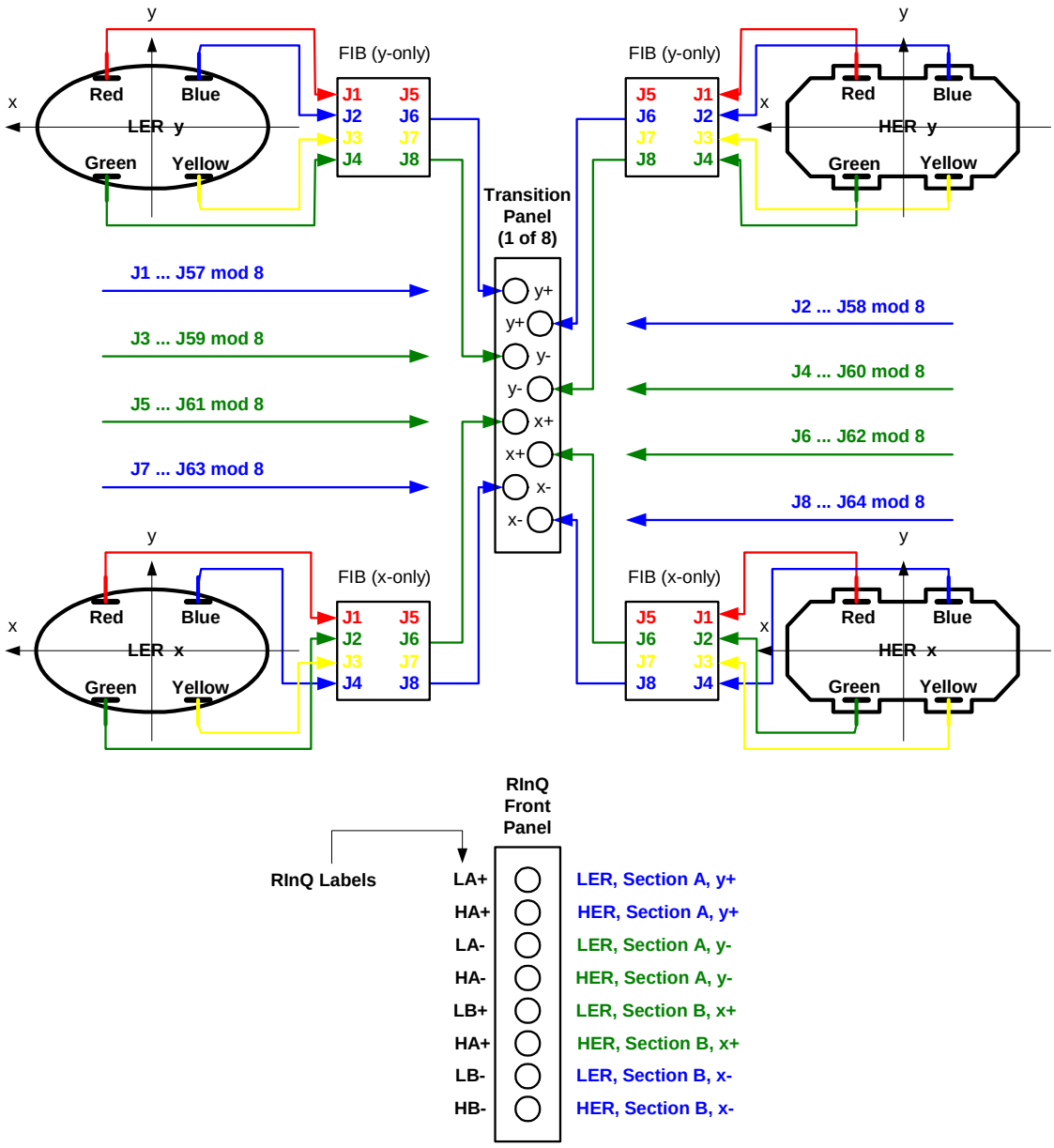

Figure 2.2 BPM system layout (x-only and y-only BPMs). Note: The beam (z+) goes into the page for HER and out of the page for LER.

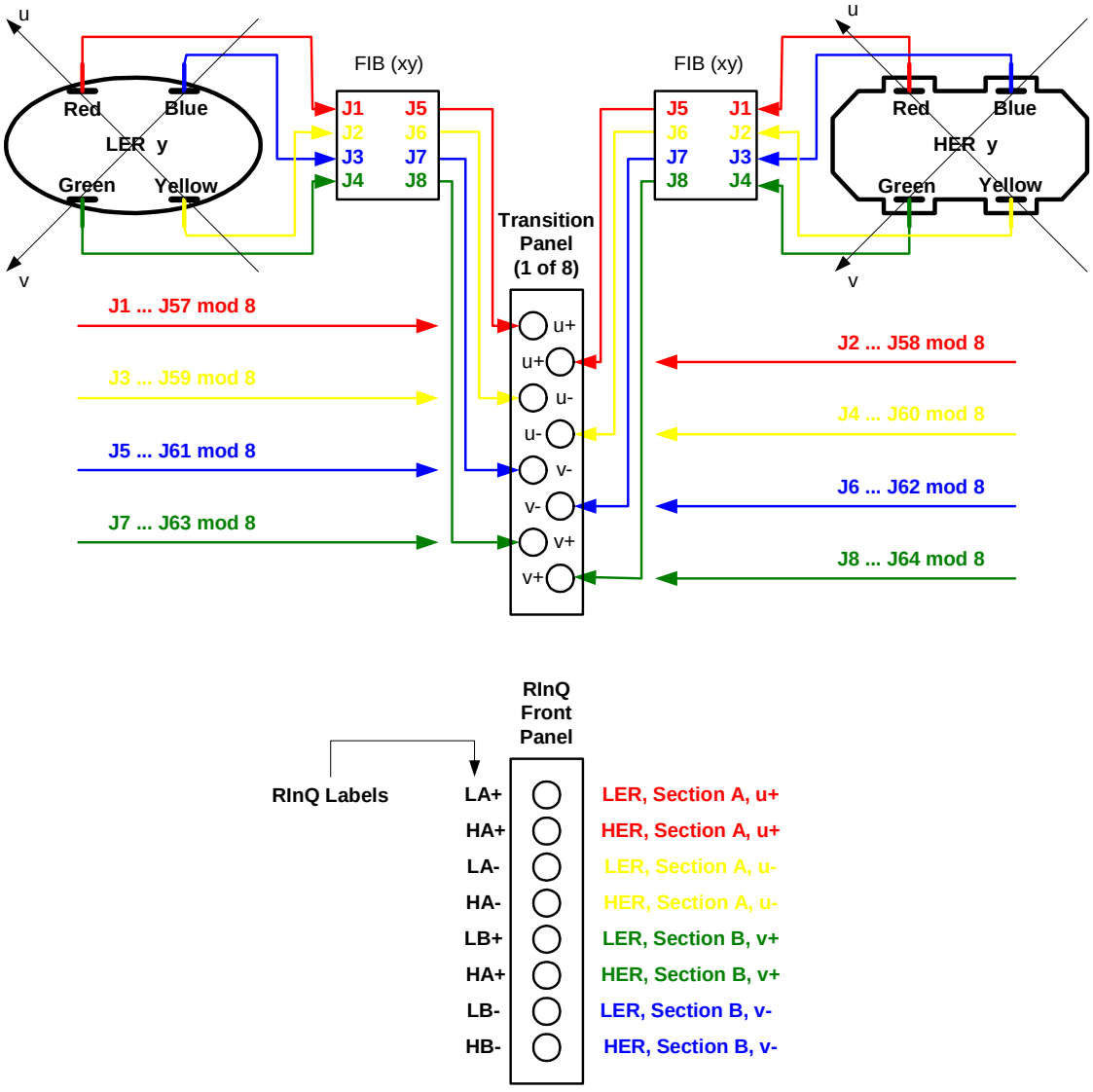

Figure 2.3 BPM system layout (xy BPMs). Note 1: The beam (z+) goes into the page for HER and out of the page for LER. Note 2: Blue and green are swapped at the RInQ.

3. Requirements and functionality

3.1. Resolution

The BPM requirements for PEP-II are listed in Table 3.1.

Table 3.1. Requirements.

Bunches	Passes	Number of particles per bunch	Resolution (μm)
Single	1	$5x10^{8}$	1000
Multi	1024	$1x10^{10}$	15

3.1.1. Thermal noise

The thermal noise power density generated by a resistor at temperature T (degree K) over the measurement bandwidth B (Hz) is (in an open circuit)

$$P_n(W) = 4kTB$$

with k = Boltzmann constant. When the circuit is terminated into a resistor of the same value

$$P_n(W) = kTB$$

which corresponds to

$$P_n(dBm) = -174 + 10\log(B).$$

In this case the source is capacitive but the cable is lossy enough to behave as a 50 Ω source. For B = 10 MHz

$$P_n(dBm) = -104$$

or

$$V_n(\mu V) = 1.4.$$

The instantaneous beam current is given by

$$I_b(t) = \frac{Ne}{\sigma\sqrt{2\pi}}\, e^{-t^2/2\sigma^2}$$

where Ne is the charge and σ is the bunch length. The cosine series for a pulse train with bunch spacing T is

$$I_b(t) = \frac{Ne}{T} + 2\frac{Ne}{T}\sum_{m=1}^{\infty} e^{-\frac{(m\omega_0\sigma)^2}{2}} \cos(m\omega_0 t)$$

where $\omega_0 = 2\pi/T$. The beam current m frequency component is

$$I_b(m\omega_0) = 2\frac{Ne}{T}$$

if the bunch length is short compared with the wavelength. The voltage at the connector is

$$V_{rms} = \frac{2}{\sqrt{2}}\frac{Ne}{T} CplZ$$

where Z is the load impedance and Cpl is the attenuation factor due to the button capacitive coupling. In this case the 952 MHz signal component into 50 Ω at the connector for T = 1/(238 MHz), σ = 30 ps, and Cpl = 1/50 at Ne = $1x10^{11}$ ppp is

$$V_{rms} = 5.4\, V$$

which corresponds to

$$P = 25\ dBm.$$

The situation for a single bunch is a little different, since the 952 MHz signal is generated by sending the beam through a bandpass filter with 10 MHz bandwidth. In this case the signal is attenuated by 21 dB with respect to the multibunch case. The maximum 952 MHz signal component of the generated burst into 50 Ω at the connector for $5x10^8$ ppp is

$$V_{rms} = 1.8\, mV$$

which corresponds to

$$P = -42\ dBm.$$

From the signal and estimated noise figure the expected resolution can be calculated. The position is calculated from the difference over sum of the signal into the positive and negative channels, i.e.,

$$x = \frac{b}{2}\frac{A-B}{A+B}$$

with b = 33 mm. The resolution is given by

$$\sigma_x = \frac{b}{4}\left(\frac{N}{S}\right)$$

where S/N is the voltage signal to noise ratio. The estimate noise figure is given in Table 3.2.

Table 3.2. Estimated noise figure.

Item	Gain (dB)	NF (dB)	Description
Bandpass filter	-1	1	Removes other harmonics
Combiner	3	-3	Combines two buttons
Cable	-7.2	7.2	Maximum attenuation
Coupler	-1.4	1.4	Calibration coupler
Attenuator	-4	4	Attenuator
Bandpass filter	-1	1	High Q generates SB ringing
Attenuator (min)	-1.6	1.6	Attenuator for dynamic range
RF amplifier	18	5+3	Improve NF and adjusts SB amplitude
Switch	-2.6	(2.6)	Two switches for mux
Attenuator	-7	(7)	Attenuator
I&Q demodulator	-10.5	(10.5)	Downconvert to baseband
IF amplifier	13	(1)	Adjusts the voltage for T/H
Track and hold	1	(10)	Track and hold
Op-amp	2	(2)	Optimize the ADC's dynamic range
Total	0.7	24.2	

Thus for single bunch and single pass with N = $5\text{x}10^8$

$$\sigma_x = 150\,\mu m$$

and for multi bunch and single pass with Ne = $1\text{x}10^{11}$

$$\sigma_x = 0.06\mu m.$$

3.1.2. Electronics

Other resolution limiting factors are track-and-hold noise and ADC resolution. The specified hold noise is

$$V_{rms} = 60\mu V$$

and the ADC's resolution due to the LSB is given by the 10-V dynamic range divided by the 2^{14} number of counts while the rms error is given by the LSB divided by the quantization error

$$V_{rms} = \frac{LSB}{\sqrt{12}} = 175\mu V.$$

The total noise due to the combined factors is given in Table 3.3.

Table 3.3. Position resolution from noise effects.

Type of noise	V_{rms}(μV)	Position resolution	(μm)
		Ne = $5x10^8$ SB	Ne = $1x10^{11}$ MB
Thermal	60	150	0.06
T/H Hold	60	150	0.06
ADC	175	440	0.17
Total	195	490	0.19

3.1.3. Electronics reflection

An additional limiting factor for the single bunch in the gap is given by the electronics reflection coefficient. The BPM button in fact is a highly reflective source and the measurement accuracy of the pilot bunch is impaired by the multiple reflections through the cable. The problem is illustrated in Figure 3.1.

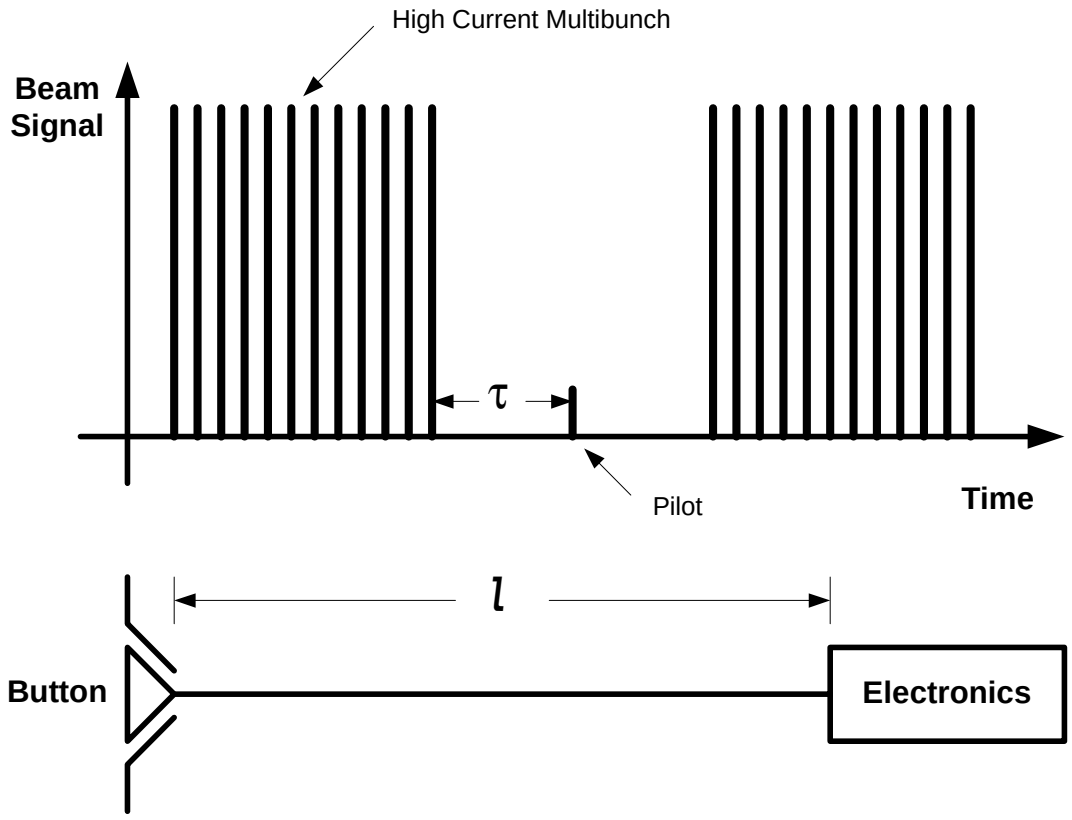

Figure 3.1 A pilot buch to be measured in the multibunch abort gap.

This problem was addressed and solved from two different sides: the electronics is designed to provide a good load and an isolator is included to reduce reflection from the electronics to the button.

The effect of the reflections is shown in Table 3.4.

Table 3.4. Effect on resolution due to reflections.

Item	HER Losses (dB)	LER Losses (dB)
Cable insertion loss	-7.5	-6
Processor return loss	-20	-20
Isolator return loss	-20	-20
Cable insertion loss	-7.5	-7.5
Total return loss	-55	-42
Total resolution	150 μm x ratio	220 μm x ratio

The various contributions add up to different values for the LER and HER because of the different cable insertion loss with round trip time equal to the gap length. The total

resolution is given in units of μm x ratio, e.g., the HER the resolution is 150 μm if the single bunch and multi bunch have the current or 1.5 mm if the single bunch current is 10% of the multi bunch.

3.2. Functionality

The functionality of the BPM system is primarily set by the software but the processor module (RInQ) does have a DSP on board which can support a variety of functions. There are three basic modes of operation: single-shot executing at 120 Hz, N-turn at the 7-μs rate, and beam abort mode (2000 measurements) at the 7-μs rate. For each mode the processor will return X, Y, and TMIT while for the multi turn modes it also returns σ_X, σ_Y, and σ_{TMIT}. Each of these modes can be used for single or multi bunch (the processor will contain calibration parameters for each).

4. Filter-Isolator Box

The BPM filter-isolator box (FIB) is a passive microwave instrumentation network which performs a filtering, combining, and impedance matching function to broadband, high-frequency signal energy. The input energy sources are 30-ps impulses with peak amplitudes exceeding 500 V.

The FIB consists of traveling wave directional bandpass filters (striplines), constant-resistance low pass filters, hybrid 2-way combiners (Wilkinson), ferrite isolators, and DC bias circuitry. The center frequency is 952 MHz with a passband of 150 MHz. Out of band power is dissipated by 50 Ω high power resistors. A DC bias of up to 350 V can be put on the BPM buttons to assist in ion clearing.

Requirements and typical performance of a FIB are listed in Table 4.1.

Table 4.1. Filter-Isolator Box requirements and typical performance.

Parameter	Requirement	Performance (typical)
Center frequency (MHz)	952 (nominal)	942.6
Bandwidth @ 3 dB (MHz)	50 < BW < 238	150
Insertion loss @ 952 MHz(dB)	< 2	1.6
Power handling (W)	> 50	OK
Output return loss (dB)	< -19	-21.5
Input return loss (dB)	< -6	-17.9
Bias voltage (V)	350	OK

A photograph of the FIB assemblies are shown in Figure 4.1 and a functional diagram is shown in Figure 4.2. Circuit diagrams of the FIB are shown in Figures 4.3-5.

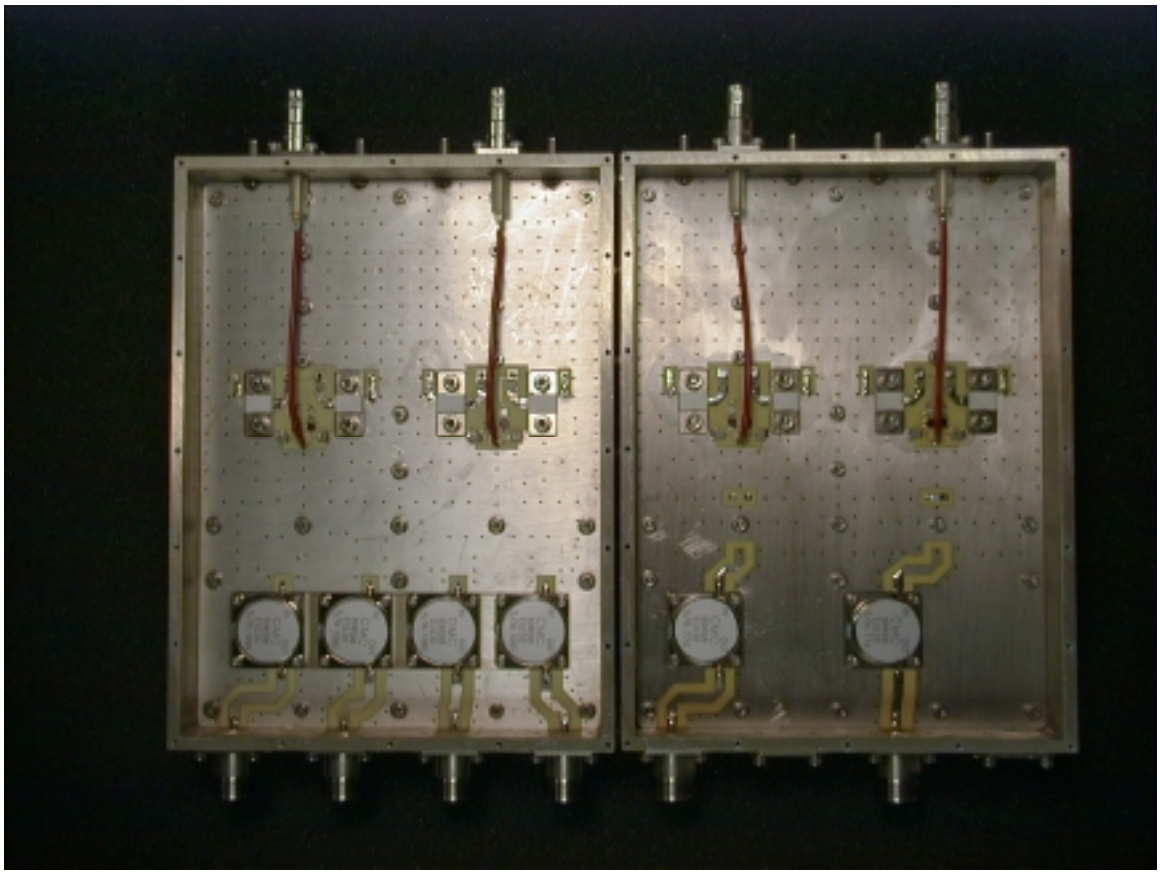

Figure 4.1 Filter-Isolator Box. Left: XY (4x4). Right: X or Y Only (4x2).

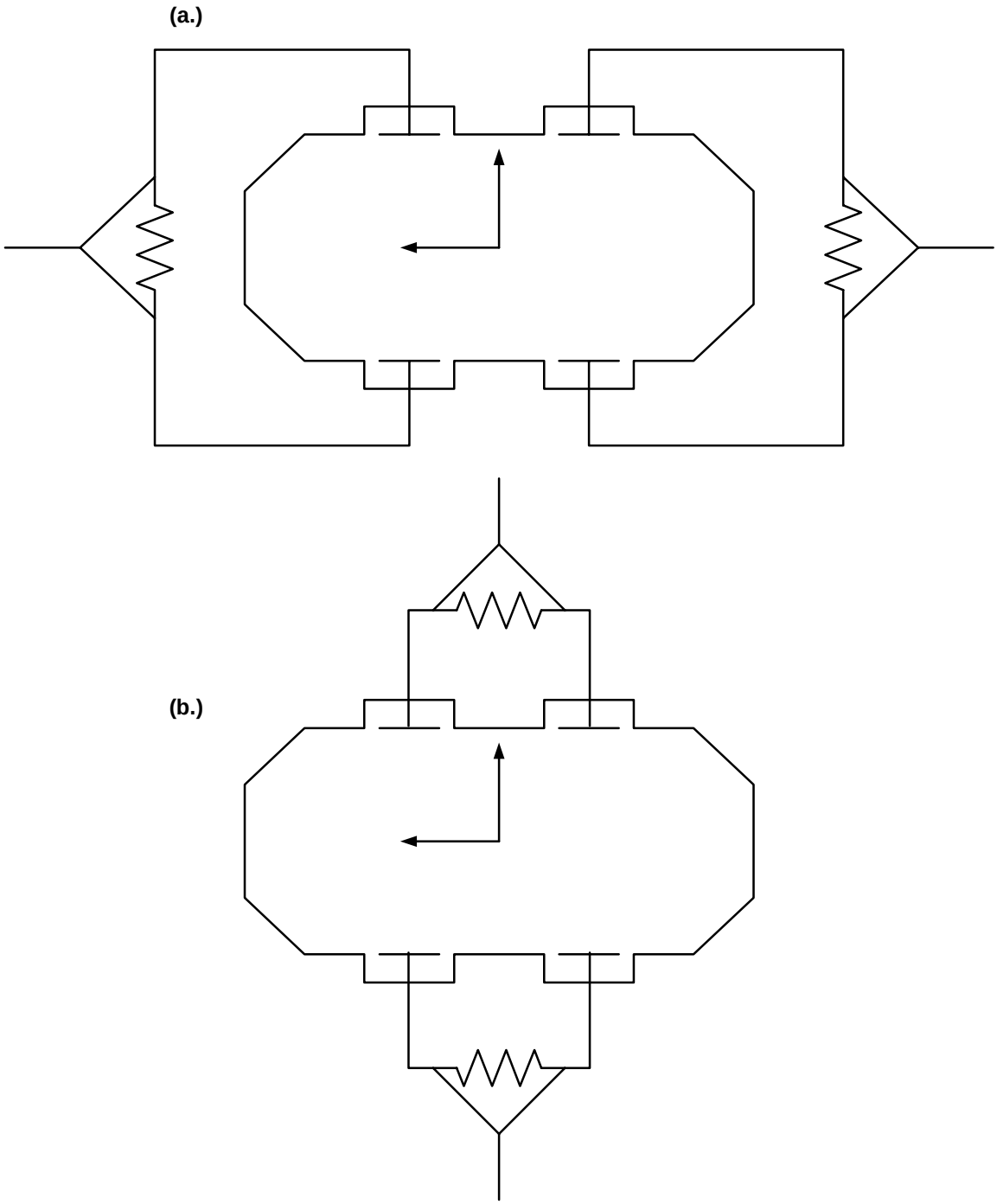

Figure 4.2 Button combinations: a.) X-only and b.) Y-only

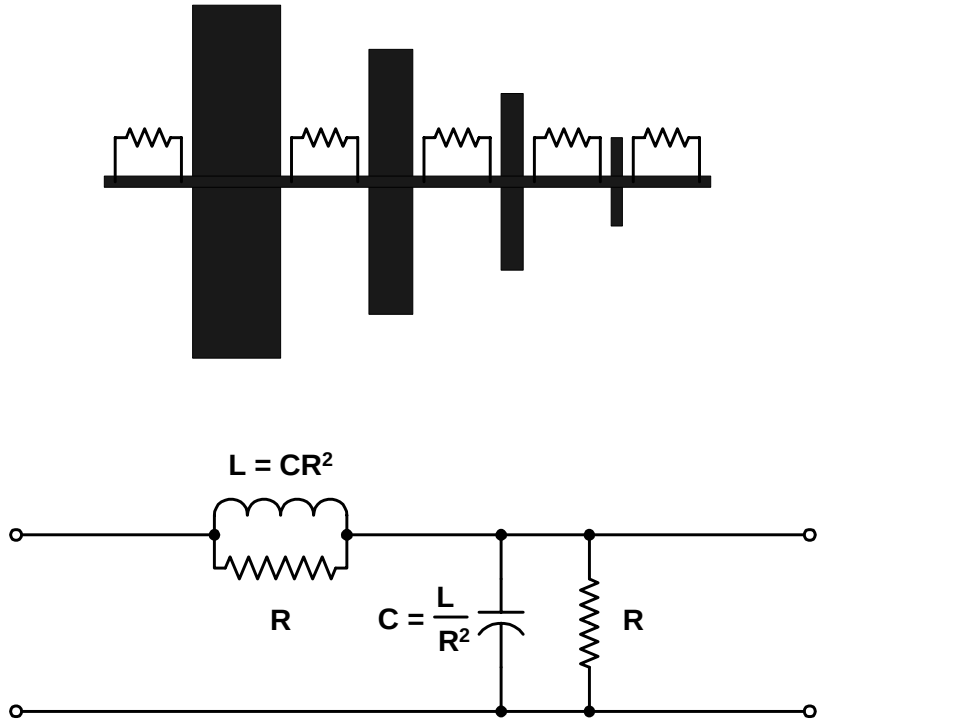

Figure 4.3 Low pass filter.

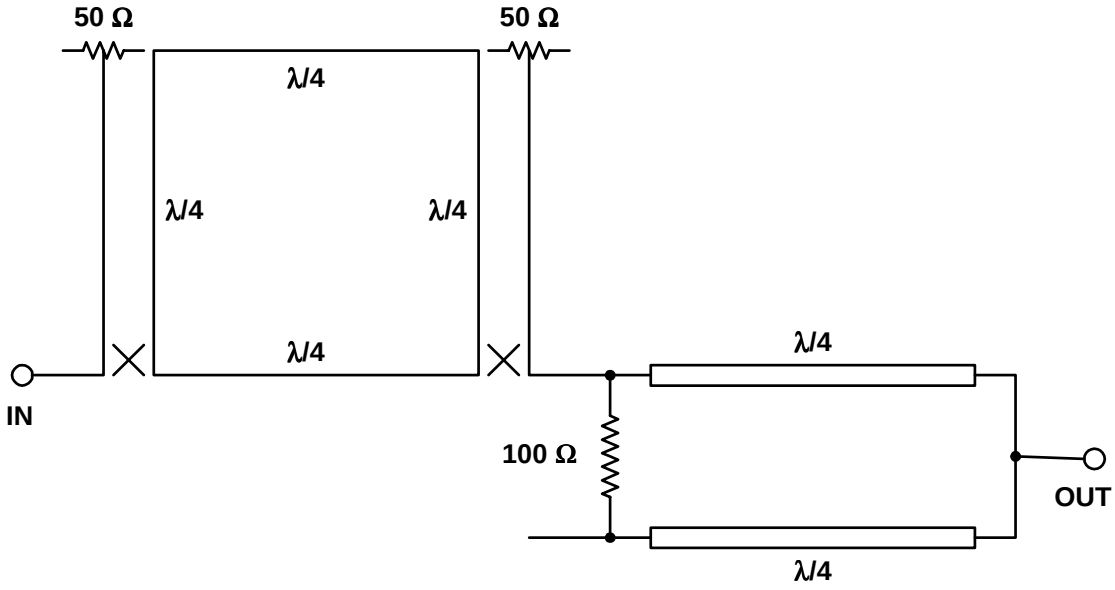

Figure 4.4 Bandpass filter and Wilkinson combiner.

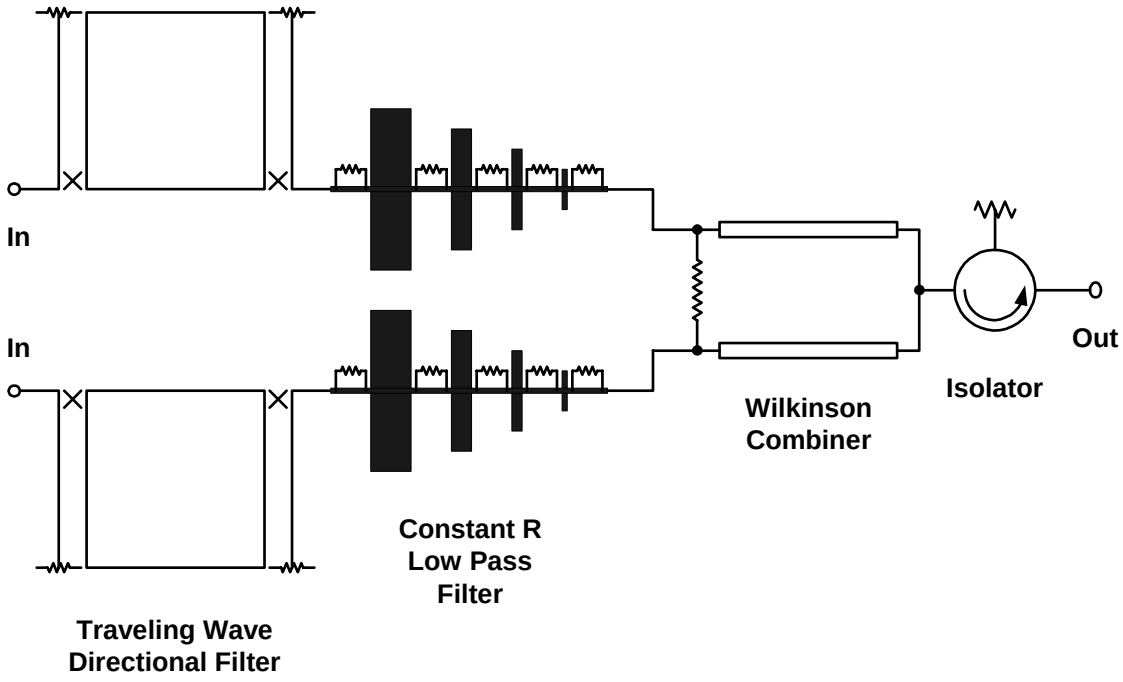

Figure 4.5 FIB block diagram.

5. Processing module (Ring I&Q – RInQ)

The in-phase and quadrature (I&Q) processing is based on baseband conversion using I&Q demodulators. The amplitude of the demodulated signal is digitally calculated by

$$V_0 = \sqrt{V_I^2 + V_Q^2}$$

where

$$V_I = V_0 \sin\phi$$
$$V_Q = V_0 \cos\phi$$

and the position by

$$x = K \frac{(V_1 + V_2) - (V_3 + V_4)}{(V_1 + V_2) + (V_3 + V_4)}$$

with K a constant determined by the geometry of the buttons.

The formulas are correct in the ideal case but when dealing with real components there are several sources of error to be considered:

1. channel gain mismatch
2. channel offset mismatch
3. amplitude and phase unbalance.

Channel gain mismatch produces an error for beam off center and for different beam currents. This is calibrated by ramping the signal power at the channel inputs and measuring the slope of the transverse function. Channel offset mismatch is calibrated by measuring the channel outputs with no signal applied and compensating for the apparent beam position.

Although these two calibration procedures are applicable to all systems using linear processing, phase and amplitude unbalance are typical of the rf processing method chosen for PEP-II. The previous formulas are valid only for an ideal system, when the quadrature phase is exactly 90 deg and both channel amplitudes are the same. The calibrator must measure the amplitude unbalance and the phase unbalance in order to compensate for these errors as well.

A photograph of a RInQ module is shown in Figure 5.1 and a block diagram of the module is shown in Figure 5.2. The rf I&Q demodulator provides demodulated signals to the baseband processor that preprocesses the information to be transmitted to the control system. The calibrator, in conjunction with the local oscillator, permit the processor to be calibrated for offset and gain errors and vector errors.

The rf processor measures the position both for multi bunch, in narrowband mode, and for single bunch, in wideband mode, where the rf burst is generated by the ringing when excited by an impulse. In the latter case the calibrator must sweep through the frequency band, in order to map all the bandwidth.

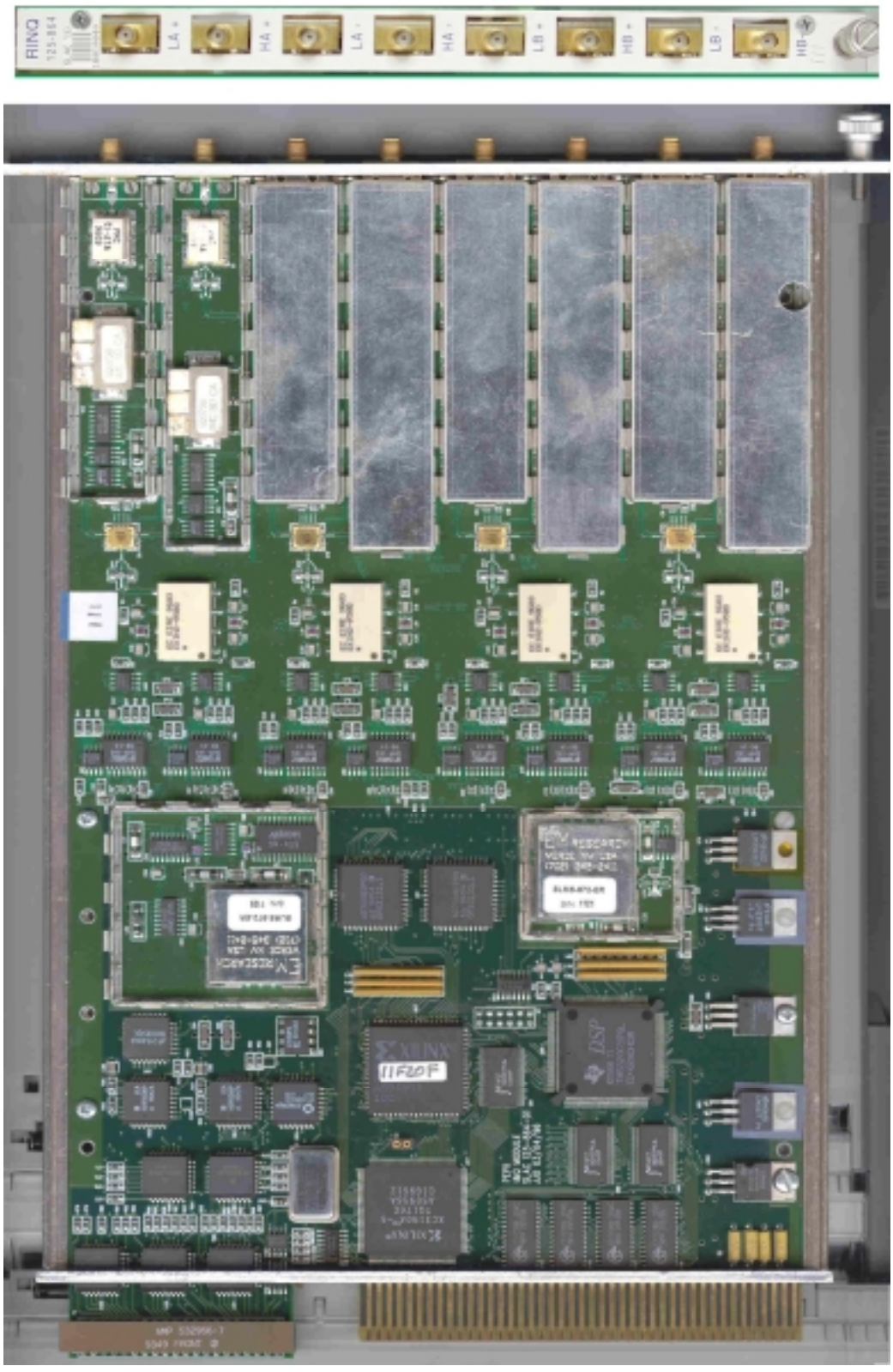

Figure 5.1 RInQ Module.

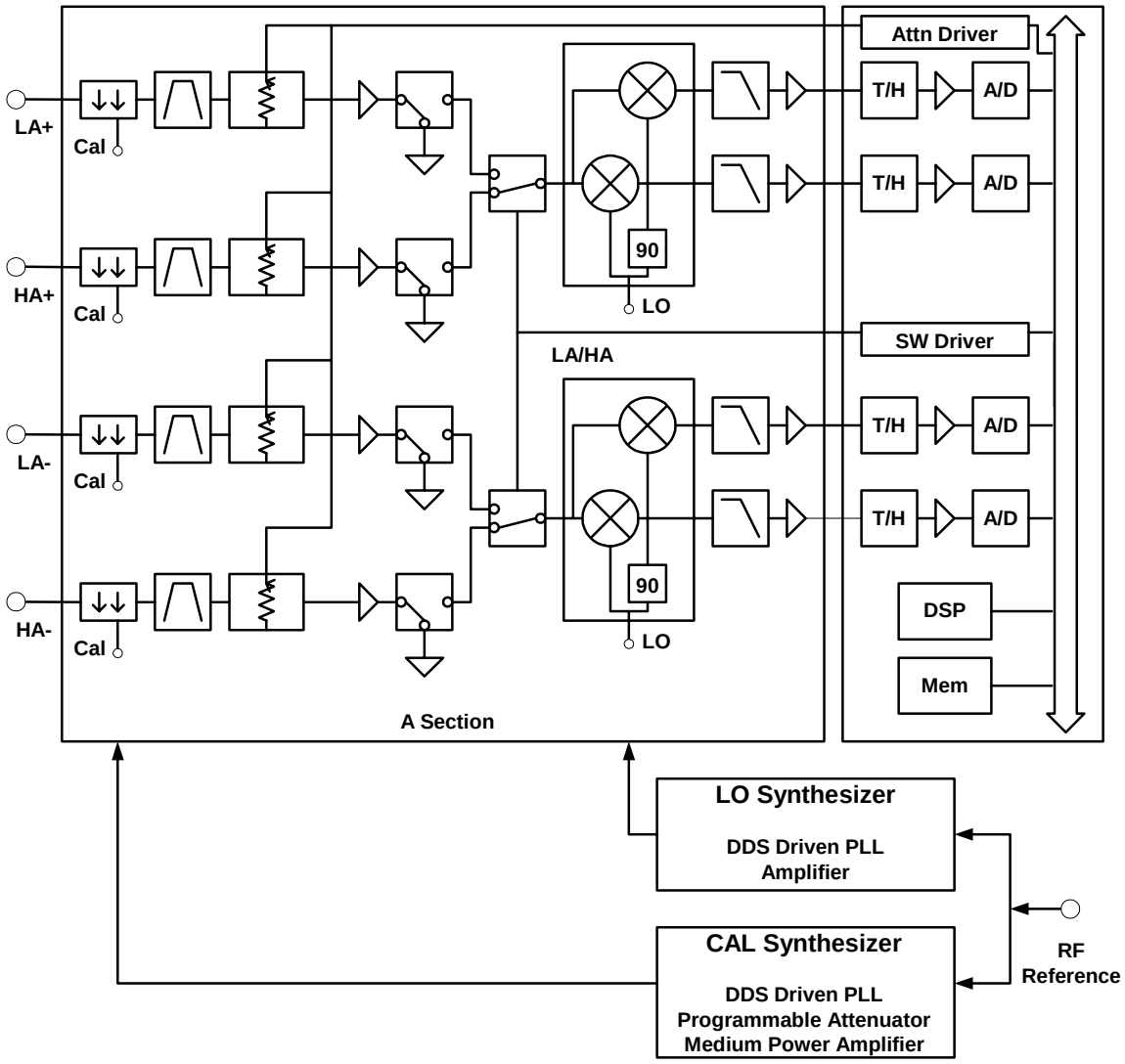

Figure 5.2 Ring RF I&Q Demodulator (RInQ). Note: only section A is shown; section B is identical.

5.1. RF I&Q demodulator

There are two modes of operation: narrowband and wideband. The narrowband case is used when the multi bunch signal must be measured. The wideband case is used when the single bunch must be measured. In the wideband case there is no rf carrier present, but the rf burst is generated by the single pulse ringing through the bandpass filter.

5.1.1. Requirements and functionalities

The requirements are listed in Table 5.1.

Table 5.1. RF I&Q demodulator requirements.

Item	Value
RF carrier (MHz)	952
Maximum input level (dBm)	+28
Minimum input level (dBM)	-51
Minimum SNR (dB)	19
System bandwidth (MHz)	10
Sampling rate (kSa/s)	140
Linearity	0.1 dB
Channel matching	0.1 dB
Chan-to-chan isolation (dB)	86

The maximum expected input level is in the multi bunch case, for 10^{11} ppp, and the beam 1 cm off center on the electrode side, where the required resolution is 15 μm. The minimum input is in the single bunch case, for $5x10^{8}$ ppp, and the beam 1 cm off center on the electrodes opposite side where the required resolution is 1 mm.

The channel-to-channel isolation requirement comes from the fact that two BPMs, one from HER and one from LER, are multiplexed together. It could happen that a single bunch (with low current) from one machine, when the other machine is running at full current. In this case the signal-to-interference should be kept as low as the signal-to-noise ratio to achieve the required resolution.

5.1.2. Block diagram and circuit characteristics

The BPM rf processor block diagram is shown in Figure 5.3. The coupler accepts the calibration signal, the BPF selects the 952-MHz component, an attenuator and amplifier extend the dynamic range of the circuit, the switch multiplexes HER and LER, the I&Q demodulator and low pass filter amplitude demodulates the input signal, and the T&H acquires the signal to be digitized by the ADC.

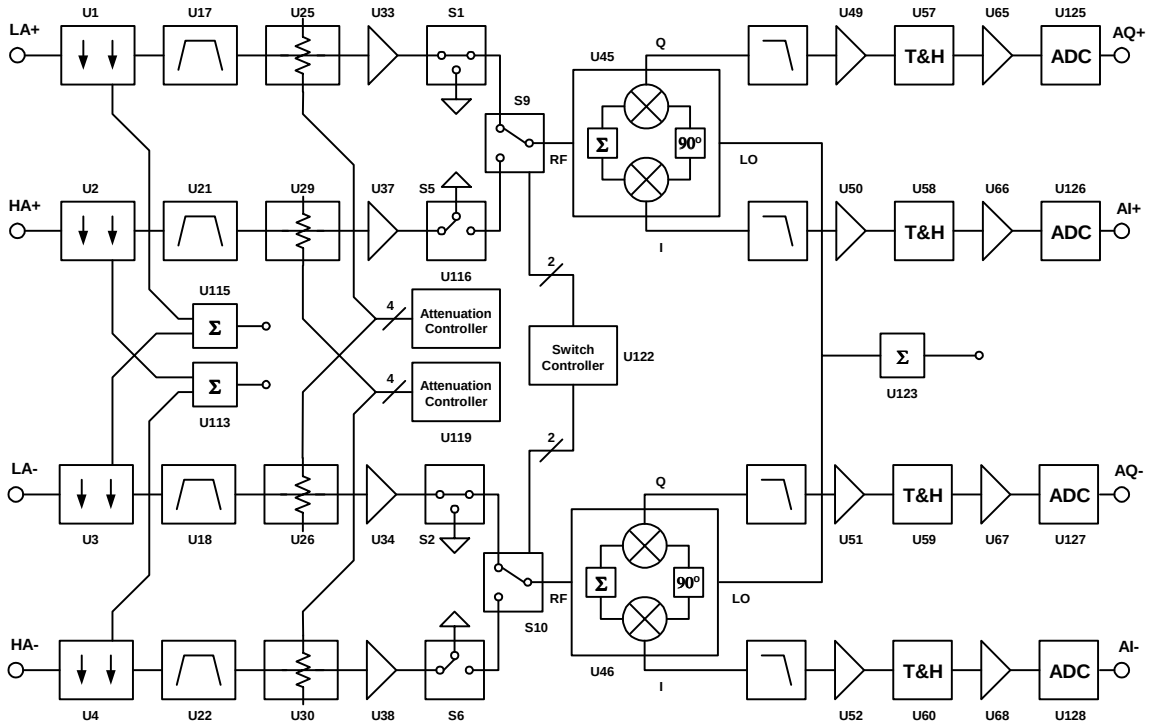

Figure 5.3 BPM processor block digram (A section).

5.2. Calibrator

5.2.1. Requirements and functionalities

The BPM accuracy requirements specify the error to be smaller than 0.25 mm rms, which corresponds to 0.2 dB in the worst case, when the sensitivity is 0.9 dB/mm. The calibrator must be able to measure the electronics errors in order to compensate for these effects. By giving the same weight to amplitude and phase unbalance and channel mismatch, the actual requirements are listed in Table 5.2.

Table 5.2. Calibration parameter requirements.

Item	Error
Channel offset mismatch (dB)	0.03
Amplitude unbalance (dB)	0.08
Quadrature phase unbalance (deg)	1
Channel gain mismatch (dB)	0.08
RMS (dB)	0.12

Offset mismatch, also called pedestal, is measured by turning the calibrator off and measuring the apparent beam position.

Gain mismatch is measured by sweeping the calibrator signal power and fitting the single channel transfer function to the best line.

Phase and amplitude unbalance are measured by setting the calibrator frequency to a few kHz (i.e., 70 kHz) off the local oscillator frequency, digitizing at the maximum sampling rate (134 kHz) and then running the amplitude and phase calculation algorithm. This is a fixed frequency curve fitting algorithm, which solves a system of linear equations for amplitude phase and offset.

Since the calibrator must be able to perform also when the beam is present, it works at a frequency different from the beam fundamental. When it is turned on, the local oscillator, which provides the reference frequency to the rf processor for the baseband conversion, is also detuned from the beam frequency. This is particularly important for pedestal calibration when no signal must be applied to the inputs.

Calibration is performed during the gap. The residual beam frequency component is expected to be at –25 dBm in the worst case (30 dBm maximum power and 55 db rf processor return loss. The minimum calibration power is expected to be –40 dBm (21 dBm maximum power and 60 dB programmable attenuation. A 3-pole digital filter can be applied to the digitized data to knock the residual beam signal component to 40 dB below the calibration signal, even though the calibration algorithm is expected to be insensitive to other frequencies.

The wideband mode is calibrated by taking different measurements over the working bandwidth, 932 MHz to 972 MHz.

5.2.3. I&Q processing errors

The digitized values are

$$V_I' = V_{I0} \sin\left(\phi + \frac{\varepsilon}{2}\right)$$
$$V_Q' = V_{Q0} \cos\left(\phi - \frac{\varepsilon}{2}\right)$$

where V_{I0} and V_{Q0} include the amplitude unbalance and ε is the quadrature phase error. First correct for gain and offset mismatch

$$V = g(V' - p)$$

where g is the channel gain and p the channel pedestal. The result is still affected by quadrature phase unbalance.

$$V_I' = V_0 \sin\left(\phi + \frac{\varepsilon}{2}\right)$$
$$V_Q' = V_0 \cos\left(\phi - \frac{\varepsilon}{2}\right)$$

These formulas, for small phase unbalance, can be approximated by

$$V_I' = V_0\left(\sin\phi\cos\frac{\varepsilon}{2} + \cos\phi\sin\frac{\varepsilon}{2}\right) \cong V_0\left(\sin\phi + \frac{\varepsilon}{2}\cos\phi\right)$$
$$V_Q' = V_0\left(\cos\phi\cos\frac{\varepsilon}{2} + \sin\phi\sin\frac{\varepsilon}{2}\right) \cong V_0\left(\cos\phi + \frac{\varepsilon}{2}\sin\phi\right)$$

and

$$V_I' = V_I + \frac{\varepsilon}{2}V_Q$$
$$V_Q' = V_Q + \frac{\varepsilon}{2}V_I.$$

Inverting them

$$V_I = V_I' - V_Q'\frac{\varepsilon}{2}$$
$$V_Q = V_Q' - V_I'\frac{\varepsilon}{2}$$

and the channel amplitude can be derived from the sum of squares of these components.

5.2.2.1. Consequences of amplitude unbalance

For the case of V_1 not equal V_2 but $\Delta\phi$ equal zero the calculated amplitude is

$$V = M(t)\sqrt{\left[V_1\cos(\omega_{IF}t + \varphi)\right]^2 + \left[V_2\sin(\omega_{IF}t + \varphi)\right]^2}$$

$$V = M(t)\left[\left(\frac{V_1 + V_2}{2}\right) + \left(\frac{V_1 - V_2}{2}\right)\cos 2(\omega_{IF}t + \varphi)\right]$$

which is a sine wave with amplitude equal to half the mismatch amplitude and twice the IF frequency. The measured position error for beam on center is given by

$$x = 10\sqrt{2}\log(e)\left(\frac{\Delta V}{V}\right)$$

i.e., for a tolerated error of 150 μm accuracy is 2.4% which corresponds to 0.2 dB uncorrected amplitude unbalance.

5.2.2.2. Consequences of phase unbalance

For the case of V_1 equal V_2 (assume for convenience unity) but $\Delta\phi$ not equal to zero the calculated amplitude is

$$V = M(t)\sqrt{[\cos(\omega_{IF}t + \varphi)]^2 + [\sin(\omega_{IF}t + \varphi + \Delta\varphi)]^2}$$

$$V = M(t)\left[1 + \left(\frac{\Delta\varphi}{2}\right)\cos 2(\omega_{IF}t + \varphi)\right]$$

which is a sine wave with amplitude equal to half the phase shift and twice the IF frequency. The measured position error for beam on center is

$$x = 10\sqrt{2}\log(e)(\Delta\varphi)$$

i.e. for a tolerated error of 150 μm accuracy the uncorrected phase unbalance is 1.5 degrees.

5.2.3. Block diagram and circuit characteristics

A block diagram of the calibrator is shown in Figure 5.4. The calibrator is based on the DDS-driven PLL technique. The reason to choose such a solution are a broad output bandwidth, small step sizes, low phase noise and spurious performance, and minimal complexity and cost. The DDS clock is derived by a prescaler working with the 119 MHz rf reference as an input. It is programmed to provide an output frequency from 9.32 MHz to 9.72 MHz in 7 mHz steps. The PLL final output frequency is 932 to 972 MHz, which is given by the feedback loop divisor that is set at 100.

The digital programmable attenuator allows the power to be swept in 4 dB steps over a 60 dB dynamic range. The medium power linear amplifier with turn off capability provides the required power to the rf processor inputs and guarantees the isolation during normal mode of operation. Switches and power splitters distribute the signal to the eight rf processor inputs. The design of the calibrator allows the amplitude and phase to be swept at different frequencies.

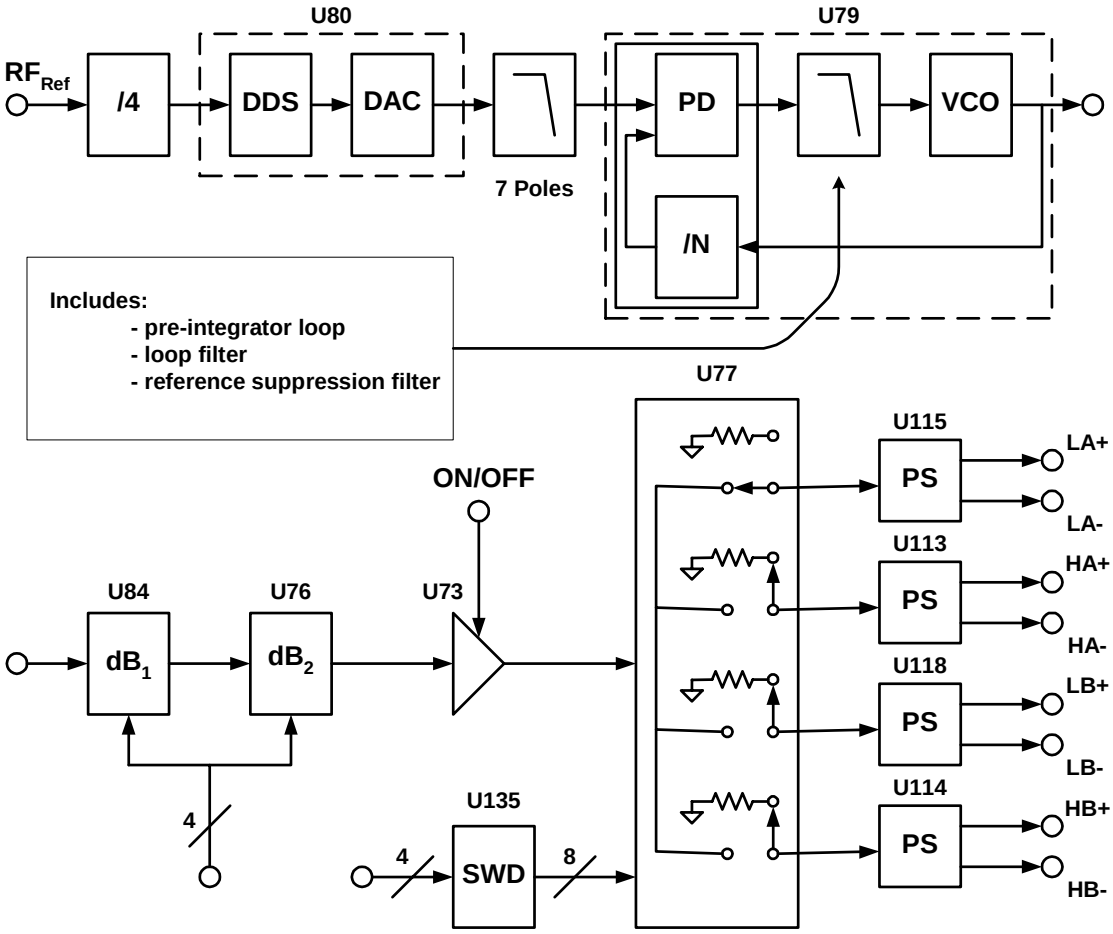

Figure 5.4 Calibrator block diagram.

The specifications for the calibrator are listed in Table 5.3.

Table 5.3. Calibrator specifications.

Item	Value
Frequency sweep (MHz)	932 – 972
Frequency steps (Hz)	0.7
Maximum phase jitter (degrees) (at 100 kHz for 100 samples)	1
Phase noise (dBc/Hz) (at 1 kHz)	-61
Spurious (dBc) (at 1 – 100 kHz)	-61
Amplitude step (dB) (0 – 60 dB)	4
Maximum power at channel input (dB)	+21
Calibrator off switch	Yes

As shown in the block diagram the prescaler provides the clock to the DDS by dividing the 119 MHz rf reference by 4. The DDS generates a 9.32 to 9.72 MHz signal that is multiplied by 100 in the PLL. The digital programmable attenuator allows the amplitude to be swept. The amplifier optimizes the power for the calibration requirements and the switch and splitters provide the distribution.

5.4. Local Oscillator

In Figure 5.5 a block diagram of the Local Oscillator (LO) is shown.

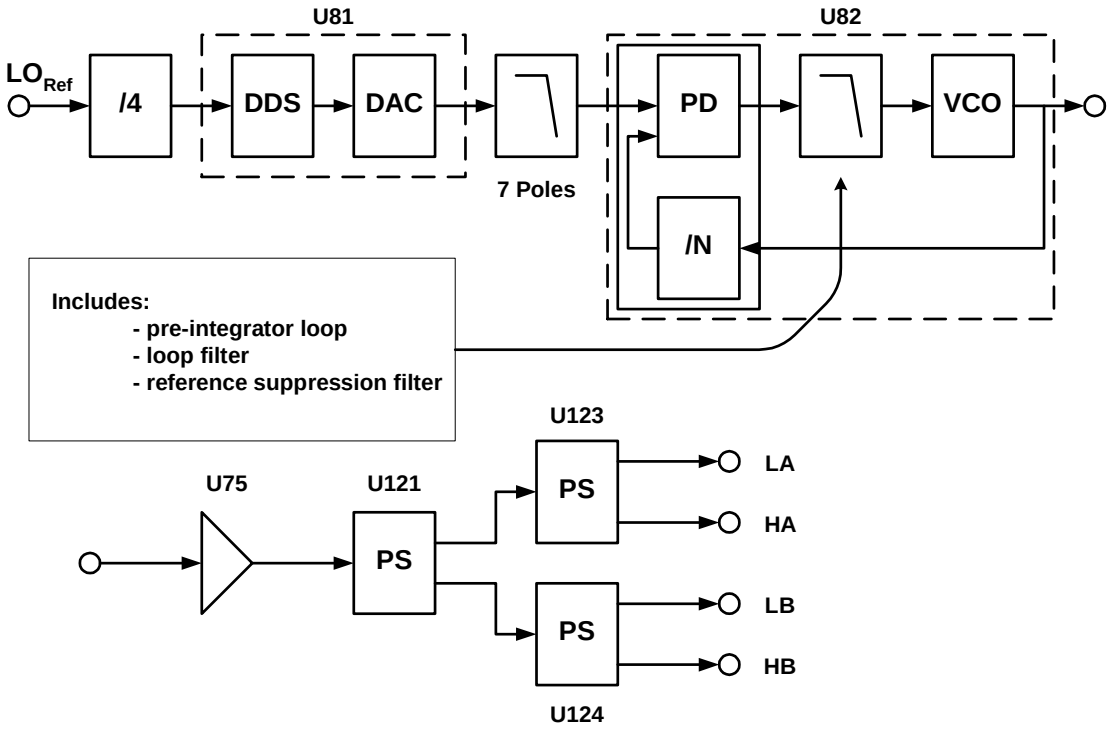

Figure 5.5 Local oscillator block diagram.

The LO is based on the same technique as the calibrator and uses the same parts except for the output amplifier. The output amplifier (non-switchable) provides +7 dBm to each of the four I&Q demodulators.

5.4. Baseband processor

The PEP-II BPM has three basic modes:

2. Single-shot measurement mode:
   a. Executes at the 120 Hz rate.
   b. Returns X, Y, TMIT via Dual-Port Memory.
   c. The location of the data is supplied by the DSP memory map.

3. N-turn measurement mode:
   a. Take N measurements at the 7 μs rate.
   b. Returns X, Y, TMIT, σ_X, σ_Y, σ_{TMIT} via Dual-Port Memory.
   c. The location of the data is supplied by the DSP memory map.

4. Beam abort mode:
   a. Take N measurements at the 7 μs rate.
   b. Take a maximum of 2000 measurements.
   c. Returns X, Y, TMIT, σ_X, σ_Y, σ_{TMIT}, and raw data via Dual-Port Memory.
   d. The location of the data is supplied by the DSP memory map.
   e. This data is not overwritten by the single-shot or N-turn modes.

A block diagram of the digital hardware of the baseband processor is shown in Figure 5.6.

5.4.8. DSP Program and Data Memory (32k x 32)

The DSP Program and Data Memory is 32k x 32. The DSP uses the same memory for program and data. When the DSP boots, data from the Boot Memory is transferred into the Program and Data Memory. The remaining memory can be used for data storage. From CAMAC this memory is 16-bits wide.

5.4.8. Boot Memory (32k x 16)

The Boot Memory is used to store the power-up code for the BPM as well as any necessary constants. Both CAMAC and the DSP have access to the EEPROM. Care must be taken when writing to the EEPROM since it is a very slow device. From the DSP, a software algorithm must be implemented to insure a 10 ms write time for a single word.

When the DSP is reset, either on a power-up or with an F9A2 command, data from the Boot Memory is moved from the slow speed EEPROM to faster Program and Data Memory.

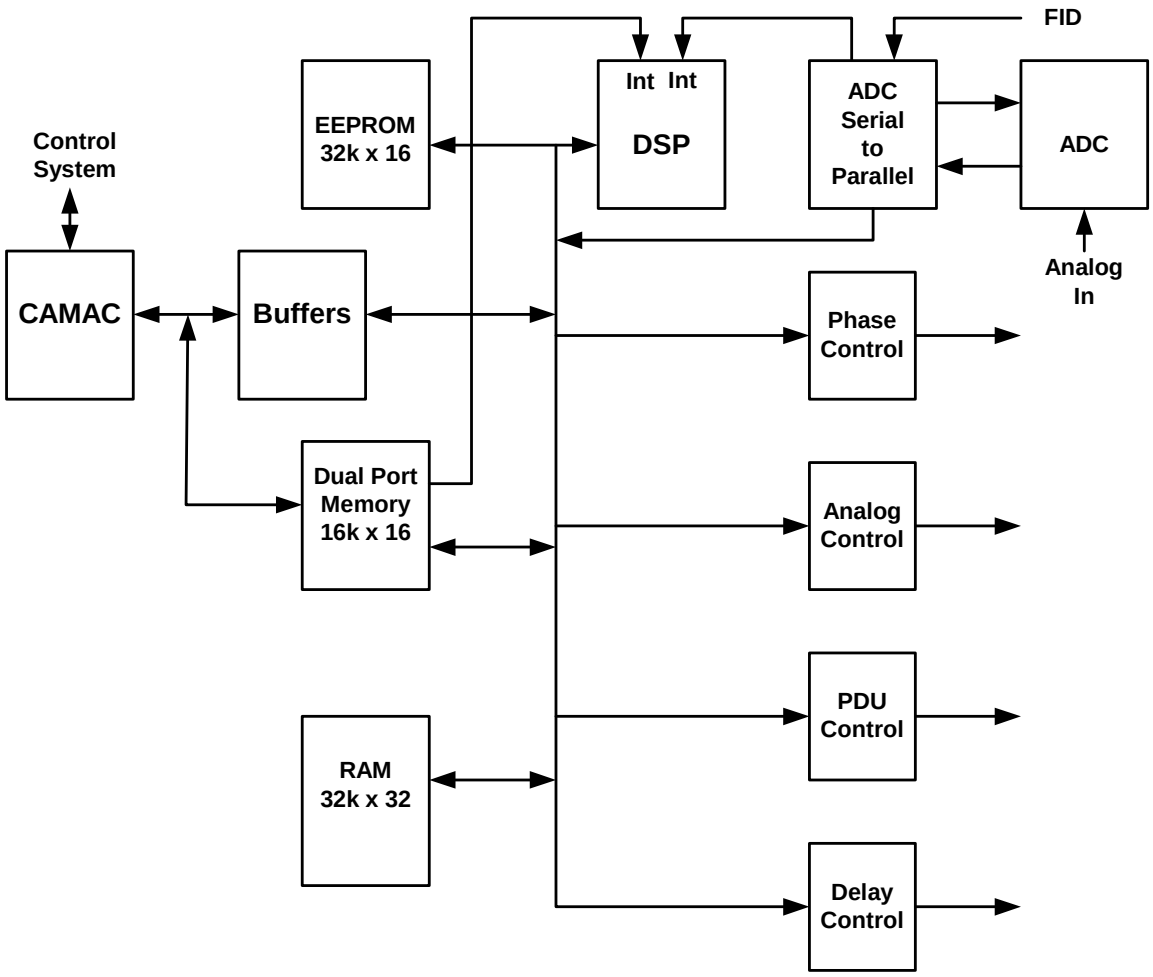

Figure 5.6 RInQ digital block diagram.

To download a new program to the EEPROM the DSP must be in the HOLD mode, F9A3. A block of data consisting of a starting address and 64 words of data are written to the BPM via CAMAC. The address of the block must start on a 64 word boundary, A0 through A5 equal zero, and each word must be written within 100 μs of each other. The data is transferred to the EEPROM starting at the supplied address using page-mode write cycles. After the 64 word block has been transferred to the EEPROM, it can take up to 10 ms to finish writing the block into memory. During this time no more data can be sent to the EEPROM. To determine when the BPM is ready for another block of program memory, CAMAC can either wait 10 ms or use the DATA Polling feature of the EEPROM. In this mode, CAMAC reads the last address written. When the data read compares with the data written, the EEPROM block write is finished. Once the download operation is complete a reset, F9A2, will restart the DSP and execute the new program. Single word transfers to and from the EEPROM are permitted and the completion of a write must be insured by waiting 10 ms or DATA Polling as described earlier.

To change the Boot Code CAMAC must first gain access to the Boot Memory by issuing an F29A12 to unlock the module, then an F9A3 to set the HOLD mode. The data in the Boot EEPROM can be altered. When new code has been loaded, CAMAC must boot the DSP by issuing F9A2. This will also release the HOLD state set by the F9A3.

5.4.8. Dual Port Memory (16k x 16)

The Dual Port Memory is accessible from both CAMAC and the DSP at the same time. This memory can be used for communication between the DSP and CAMAC. Data available in the Dual Port Memory depends on the mode the BPM is in. The location of the data is determined when the DSP Code is compiled and must be known to the rest of the system for data readout. The following data is passed between the DSP and CAMAC through Dual Port Memory:

1. X
2. Y
3. TMIT
4. Status
5. σ_X
6. σ_Y
7. σ_{TMIT}
8. Good Measurement
9. Version Number of Firmware
10. Module R.N.
    a. R[8…1] Module Revision Number
    b. R[16] Production Flag
        i. 0 – Production Module
        ıı. 1 – Development Module
11. Module S.N.
    a. R[10…1] Module Serial Number
    b. R[16…11] Module Type
12. Beam Abort Data
13. Current Meas Prep Number
14. Phase Control
15. CAL Control
16. Mode Control
    a. Attenuator Setting
    b. Mux Setting
    c. Bandwidth Select

Single word messages can be passed between CAMAC and the DSP. When CAMAC writes to address 0x3FFF an interrupt is generated to the DSP (INT1). This interrupt is cleared when the DSP reads this location. If interrupts are not required, this message passing can also be done in a polled fashion. The interrupt bit is also connected to one of

the DSP Flag In bits (XF1). The DSP can poll this flag. The DSP can also clear this address and then poll it to see if CAMAC has written a non-zero value.

5.4.8. ADC Interface

The timing logic generates a Start signal to the ADC interface. The ADC interface then issues a Start Convert to the ADCs and reads the ADC out, converting the serial data to parallel. When the read out is complete, the ADC will generate an interrupt to the DSP (INT0). The interrupt is reset when the DSP reads any of the ADC values in memory. The interrupt also sets a bit in the DSP Flag In (XF0) that can be polled if interrupts are not required. The ADCs data is available in the DSP memory space.

5.4.8. DSP

Once the BPM is configured by CAMAC, the DSP will do all of the data processing and store the results in the appropriate places. The main use of the DSP is to read the 8 ADCs, compute X, Y, and TMIT and store these results in Dual Port Memory to be read out by CAMAC. The timing system will start the ADCs. When the ADCs are finished converting, the DSP will be interrupted. The DSP will then read the 8 ADC values. The ADCs are mapped into the DSP memory space.

5.4.8. CAMAC Functions

The CAMAC block consists of decoders and registers. The registers are accessible via CAMAC operations. The CAMAC functions are:

F9A0 Initialize Module; X=1, Q=1 if module is unlocked.

F9A1 Assert Reset to the DSP; X=1, Q=1 if module is unlocked.
CAMAC can not gain access to the internal data bus.

F9A2 Release Reset to the DSP; X=1, Q=1 if module is unlocked.
Release Hold to DSP if Hold was set. This will cause the DSP to reboot from EEPROM. CAMAC can not gain access to the internal data bus.

F9A3 Assert Hold to the DSP; X=1, Q=1 if module is unlocked.
This removes the DSP from the internal data bus allowing CAMAC access to the BPM internal memory.

F9A4 Release Hold to the DSP; X=1, Q=1 if module is unlocked.
This allows the DSP to resume execution from when the Hold was asserted.

F29A12 Unlock Module; X=1, Q=1.

F29A13 Lock Module; X=1, Q=1.

F1A0 Read Memory Address; X=1, Q=1.

F17A0 Write Memory Address; X=1, Q=1.
Latch status of MesPrepQ.

F0A0 Read Dual Port Memory pointed to by Memory Address; X=1, Q=1 if last requested MesPrep has finished.
Increment Memory Address Pointer.

F16A0 Write Dual Port Memory pointed to by Memory Address; X=1, Q=1.
Increment Memory Address Pointer.

F0A1 Read Meas Prep Message; X=1, Q=1.
Read Dual Port Memory location 0x3FFF.

F16A1 Write Meas Prep Message; X=1, Q=1.
Write Dual Port Memory location 0x3FFF.
Generates an interrupt to the DSP and prevents further ADC interrupts to the DSP. ADC interrupts will be re-enabled on the next Fiducial from the CAMAC backplane. An ADC interrupt will be generated on the next PDU trigger. Clears the MesPrepQ flag. When the DSP has finished the requested MesPrep, it will write the MES_DNE bit in the status register to set the MesPrepQ flag. This flag is latched by an F17A0, Write Address, which is then used to condition Q for F0A0 reads of the Dual Port Memory.

F0A2 Read DSP Memory pointed to by Memory Address; X=1, Q=1 if module is unlocked and DSP is in the Hold state set via F9A3.
Increment Memory Address Pointer.

F16A2 Write DSP Memory pointed to by Memory Address; X=1, Q=1 if module is unlocked and DSP is in the Hold state set via F9A3.
Increment Memory Address Pointer.

F0A3 Read DSP Registers and EEPROM pointed to by Memory Address; X=1, Q=1 if module is unlocked and DSP is in the Hold state set via F9A3.
Increment Memory Address Pointer.

F16A3 Write DSP Registers and EEPROM pointed to by Memory Address; X=1, Q=1 if module is unlocked and DSP is in the Hold state set via F9A3.
Increment Memory Address Pointer.

Reading and witing the Dual Port Memory can take place any time by either CAMAC or the DSP. Normally only the DSP has access to the Boot Memory and the BPM RAM.

CAMAC can gain access to the Boot Memory, DSP RAM, and DSP Registers by first putting the DSP in the Hold state via F9A3. When the DSP responds with the Hold Acknowledge, CAMAC will be allowed access to the Boot Memory, DSP RAM, and DSP Registers. This mode is used for changing the Boot EEPROM and doing Peeks and Pokes into the DSP RAM. Since the Registers and ADC are memory mapped, this mode also allows CAMAC direct access to them.

5.4.8. Memory Map

The RInQ Module operates as a memory mapped device. All code, data, control registers, and status registers are contained in memory. The following tables contain a memory map (Table 5.4), details of control registers (Tables 5.5-12), the status register (Table 5.13) and of interrupts (Table 5.14).

Table 5.4. Memory Map.

Function Code	CAMAC	Device	DSP
F0/F16A0	0x0000 16k x 16 0x3FFF	0x0000 16k x 16 Dual Port RAM 0x3FFF	0x3F0000 16k x 16 DATA 0x3F3FFF
F0/F16A1	0x3FFF 1 x 16 MesPrep	0x3FFF 1 x 16 Dual Port RAM	0x3F3FFF 1 x 16 MesPrep
F0/F16A2 RAM	0x0000 64k x16 0xFFFF	0x0000 32k x 32 0x7FFF	0x3F8000 32k x 32 0x3FFFFF
F0/F16A3 EEPROM	0x0000 32k x 16 0x7FFF	0x0000 32k x 16 0x7FFF	0x400000 32k x 16 0x407FFF
F0/F16A3 ADC Data	0x8000 8 x 16 0x8007	0x8000 8 x 16 0x8007	0x408000 8 x 16 0x408007
F0/F16A3 Status	0x8008 1 x 16	0x8008 1 x 16	0x408008 1 x 16
F0/F16A3 Timing Select	0x8009 1 x 16	0x8009 1 x 16	0x408009 1 x 16
F0/F16A3 Timing Delay	0x800A 1 x 16	0x800A 1 x 16	0x40800A 1 x 16
F0/F16A3 Timing Trigger	0x800B 1 x 16	0x800B 1 x 16	0x40800B 1 x 16
F0/F16A3 Analog Control	0x800C 1 x 16	0x800C 1 x 16	0x40800C 1 x 16
F0/F16A3 Analog Attenuation	0x800D 1 x 16	0x800D 1 x 16	0x40800D 1 x 16
F0/F16A3 LO CMD	0x800E 1 x 16	0x800E 1 x 16	0x40800E 1 x 16
F0/F16A3 CAL CMD	0x800F 1 x 16	0x800F 1 x 16	0x40800F 1 x 16

F0/F16A3 LO Write	0x8010 1 x 16	0x8010 1 x 16	0x408010 1 x 16
F0/F16A3 CAL Write	0x8011 1 x 16	0x8011 1 x 16	0x408011 1 x 16
F0/F16A3 LO Frequency	0x8012 1 x 16	0x8012 1 x 16	0x408012 1 x 16
F0/F16A3 CAL Frequency	0x8013 1 x 16	0x8013 1 x 16	0x408013 1 x 16
F0/F16A3 LO Phase	0x8014 1 x 16	0x8014 1 x 16	0x408014 1 x 16
F0/F16A3 CAL Phase	0x8015 1 x 16	0x8015 1 x 16	0x408015 1 x 16
F0/F16A3 Analog Reset	0x8017 1 x 16	0x8017 1 x 16	0x408017 1 x 16

Warning: The address space from CAMAC 0x8000 to 0x9FFF and DSP 0x408000 to 0x408FFF is not fully decoded.

There are several control and status registers accessible by the DSP. These registers are used to set up the operating mode of the BPM. The address of these registers are defined in the DSP memory map table.

Table 5.5. Timing Select (8-bit; write only). This register is used to select which one of the 16 PDU triggers the BPM will use.

BIT	Name	Definition
3 - 0	Select CHA	4-bit number to select PDU trigger 0 – 15 for Channel A
7 - 4		Unused
11 - 8	Select CHB	4-bit number to select PDU trigger 0 – 15 for Channel B
15 -12		Unused

Table 5.6. Timing Delay (16-bit; write only). This register is used to fine tune the delay from the PDU trigger to the BPM track-and-hold. The delay time is given by

$$Delay(ns) = 6.4 + \left(N/256\right)FS$$

where N is the delay value written to the Timing Delay Register and FS = 2.5 ns.

BIT	Name	Definition
7 - 0	Delay CHA	8-bit number to set the delay for Channel A
15 - 8	Delay CHB	8-bit number to set the delay for Channel B

Table 5.7. Analog Control (16-bit; write only). This register sets various parameters in the analog section.

BIT	Name	Definition	Actual Name
3 – 0	CAL ATTEN	4-bit to set the Cal Atten level	
4	CAL LA	Enable Channel LA Calibration	CAL B act low
5	CAL HA	Enable Channel HA Calibration	CAL A act low
6	CAL LB	Enable Channel LB Calibration	CAL D act low
7	CAL HB	Enable Channel HB Calibration	CAL C act low
8	CAL ON	Turn Signal, PLL, and DDS of Calibrator ON	
9	Select LA/HA	Select Channel LA/HA	SEL B/A
10	Select LB/HB	Select Channel LB/HB	SEL D/C
15-11	Unused		

Table 5.8. Analog Attenuation (16-bit; write only). This register sets attenuators in the analog section.

BIT	Name	Definition	Actual Name
3 - 0	ATTEN LA	Set Channel LA Attenuator	ATTEN B
4 - 7	ATTEN HA	Set Channel HA Attenuator	ATTEN A
8 - 11	ATTEN LB	Set Channel LB Attenuator	ATTEN D
12 - 15	ATTEN HB	Set Channel HB Attenuator	ATTEN C

Table 5.9. LO and CAL CMD (16-bit; write only). These are registers in the Local Oscillator and Calibrator DDSs.

BIT	Name	Definition
0	DWIDTH	0 – Eight Bit Data Bus 1 – Sixteen Bit Data Bus
1	SLEEP	0 – Normal Operation 1 - Sleep
2	Unused	
3	Unused	

Table 5.10. LO and CAL Write (Assembly) Register (16-bit; write only). These are registers in the Local Oscillator and Calibration DDSs. The first write loads the upper 16-bits and the second write loads the lower 16-bits of the 32-bit Assembly Register. Transfer of the data to the Frequency or Phase Registers is completed by a write to the appropriate address. See Table 5.11.

BIT	Name	Definition	Address
0 – 15 0 – 15	FREQ0 MSB FREQ0 LSB	LO Frequency MSB LO Frequency LSB	0x8010 (First Write) (Second Write)
0 – 12	PHASE	LO Phase	0x8010
0 – 15 0 – 15	FREQ0 MSB FREQ0 LSB	CAL Frequency MSB CAL Frequency LSB	0x8011 (First Write) (Second Write)
0 – 12	PHASE	CAL Phase	0x8011

Table 5.11. Load DDS Internal Registers (16-bit; write only). The Frequency and Phase Registers of the Local Oscillator and Calibrator DDSs are loaded by writing to the following addresses. Contents of the word do not matter.

Name	Definition	Address
LO_FREQ	Load LO Frequency	0x8012
CAL_FREQ	Load CAL Frequency	0x8013
LO_PHASE	Load LO Phase Offset	0x8014
CAL_PHASE	Load CAL Phase Offset	0x8015

Table 5.12. Trigger and Analog Reset Registers (16-bit; write only). Two additional registers where a write to the address causes an action are the trigger register and analog reset register.

Name	Definition	Address
TRIGGER	Trigger Channel 15 of Both PDU Timing Channels	0x800B
RESET	Reset Both LO and CAL DDSs	0x8017

Table 5.13. Status Register (16-bit; read and write). This registers contains status information on a variety of operations.

BIT	Name	Definition
0	MESDNE	Sets the MesPrep done flag. Normally reset by a new MesPrep.
1		
2		
3	LO LOCK	Indicates that the LO PLL has locked.
4	CAL LOCK	Indicates that the CAL PLL has locked.
5	NO_119	The 119 MHz clock signal is missing. The internal 30 MHz clock is being used.
6	MIS_PDU	DSP timed out waiting for a PDU trigger.
7	OVR_PDU	PDU trigger over run.
8		
9		
10		
15-11	Unused	

Table 5.14. Interrupt definition. The column labeled BOOT is the state when the module is booted.

BIT	Name	BOOT	RUN
0	INT0	1	ADC Done
1	INT1	0	New MesPrep
2	INT2	1	Undefined
3	INT3	1	Undefined

6. Software algorithm

There are two modes to program the module: local mode through the DSP and remote mode through the CAMAC interface.

6.1. Remote mode

In remote mode the module is programmed through the CAMAC interface. It is not possible to memory map the module through CAMAC and for this reason it is necessary to set the address before each write or read operation.

6.1.3. Set the channel

There are eight switches in two sets of four to select LER or HER for section A and B. The sequence of instructions are:

F17 A0 800C	Set analog control switches.
F16 A3 Data	Data = 0(1) x 200 to select channel LA (HA) and 0(1) x 400 to select channel LB(HB). (Now HA and LA are swapped and HB and LB are swapped.)

The eight attenuators are controlled in pairs, 0 is no attenuation, F is 30 dB attenuation.

F17 A0 800D	Set analog attenuator address.
F16 A3 Data	Data = Atten x 1 (LA), x 10 (HA), x 100 (LB), x 1000 (HB). (Now HA and LA are swapped and HB and LB are swapped.)

6.1.3. Set the local oscillator

The local oscillator can be programmed for frequency and phase.

6.1.3.1. Set 16-bit mode

The sequence to set the 16-bit mode is:

F17 A0 8010	Set LO write address.
F16 A3 1	Set 16-bit mode.
F17 A0 800E	Set LO CMD address.
F16 A3 0	Dummy write.

6.1.3.2. Set frequency

The 32-bit word to set the frequency is calculated by

$$word = 2^{32} \frac{freq}{clock} \frac{1}{100}$$

where freq is the desired frequency in Hz and clock is the clock frequency in Hz. Then:

F17 A0 8010	Set LO write address.
F16 A3 Data	Data = MSB (16-bits) of the 32-bit frequency word.
F17 A0 8010	Set LO write address.
F16 A3 Data	Data = LSB (16-bits) of the 32-bit frequency word.

F17 A0 8012 Set LO frequency address.
F16 A3 0 Dummy write.

6.1.2.3. Set Phase

The 12-bit word to set the phase is calculated by

$$word = 2^{12} \frac{phase}{360} \frac{1}{100}$$

F17 A0 8010 Set LO write address.
F16 A3 Data Data = 12-bit phase word.
F17 A0 8014 Set LO phase address.
F16 A3 0 Dummy write.

6.1.3. Set the calibrator

The calibrator can be programmed for frequency, phase, amplitude, and channel.

6.1.3.1. Set 16-bit mode

The sequence to set the 16-bit mode is:

F17 A0 8011 Set CAL write address.
F16 A3 1 Set 16-bit mode.
F17 A0 800F Set CAL CMD address.
F16 A3 0 Dummy write.

6.1.3.2. Set calibrator on

The calibrator is set on with:

F17 A0 800C Set analog control address.
F16 A3 Data Data = 0(1) x 100 sets the calibrator OFF (ON).

6.1.3.3. Set frequency

The 32-bit word to set the frequency is calculated by

$$word = 2^{32} \frac{freq}{clock} \frac{1}{100}$$

where freq is the desired frequency in Hz and clock is the clock frequency in Hz. Then:

F17 A0 8011	Set CAL write address.
F16 A3 Data	Data = MSB (16-bits) of the 32-bit frequency word.
F17 A0 8011	Set CAL write address.
F16 A3 Data	Data = LSB (16-bits) of the 32-bit frequency word.
F17 A0 8013	Set CAL frequency address.
F16 A3 0	Dummy write.

6.1.3.4. Set phase

The 12-bit word to set the phase is calculated by

$$word = 2^{12} \frac{phase}{360} \frac{1}{100}$$

F17 A0 8011	Set CAL write address.
F16 A3 Data	Data = 12-bit phase word.
F17 A0 8015	Set CAL phase address.
F16 A3 0	Dummy write.

6.1.3.5. Set amplitude

A 4-bit word (Atten) sets the channel attenuation. 0 is no attenuation, F is 60 dB attenuation:

F17 A0 800C	Set analog control address.
F16 A3 Data	Data = Atten

6.1.3.6. Set channel

A 4-bit word sets the analog channel.

F17 A0 800C	Set analog control address.
F16 A3 Data	Data = 0 (all OFF), 10 (LA ON), 20 (HA ON), 40 (LB ON), 80 (HB ON). [The actual configuration is F0 (all OFF), D0 (LA ON), E0 (HA ON), 70 (LB ON), B0 (HB ON)].

6.2. Local mode

In local mode the DSP controls all RInQ functions. Communication of results is as described in Section 5 and below.

6.2.1. Address table

The following table contains the dual-port memory structure when the DSP code is running.

Table 6.1. Dual-port memory structure

Address	Type	Name	Description
		summary	
0	unsigned short	version_number	version number of running firmware
1	short	revision_number	revision number first 8 bits; 16 bit set if it is a development module
2	short	serial_number	serial number of this module
3	unsigned short	last_cal_id	cal index of last cal used
4	unsigned short	status	status of the DSP boot and init
5	unsigned short	meas_status	status of the last measurement
6	unsigned short	last_prep_id	index of the last measprep received
7	unsigned short	last_iof	copy of iof register
8	unsigned short[2]	lo_freq	lower and upper bits of lo freq word
10	unsigned short	spare	spare
11	unsigned short	analog_ctrl	analog control - muxing
12	unsigned short	timdel	timing delay register
13	unsigned short[2]	cp_freq	lower and upper bits of cp freq word
15	unsigned short	cal_analog_ctrl	analog control - calibration
16	short[2]	fill_1	spares
		crate_init	
18	unsigned short[8]	hsta_bpms	describes the four (4) bpms
26	unsigned short[2]	hsta_bpmp	describes the processor
28	unsigned short[12]	pcmm_a	for x/y width of the beam-pipe
40	unsigned short[2]	tcnst_a	for tmit calculations
42	unsigned short[2]	single_bunch_tmit	for tmit calculations
44	unsigned short	sspare	spare
45	short[2]	angle_a	angle of rotation if x/y bpm
47	short[4]	offs_a	offset in microns
51	unsigned short[2]	myspare_a	spares
53	short	lo_freq_offs_hz	lo offset during calibration in Hz
54	short	cp_freq_offs_hz	calibration freq offset in Hz
55	short	lo_freq_offset	lo offset during calibration in kHz
56	short	cp_freq_offset	calibration freq offset in kHz
57	unsigned short	dbg_flag	set to 1 if using debug timing 30 MHz
58	short	n_cal_mult	multiplier of trigger freq added to cp freq
59	short	dbg_nochk	set to 4321 to ignore the checksum
60	unsigned short	phase_gain1	phase gain for first calc step in %
61	unsigned short	phase_gain2	phase gain for subsequent calc steps in %
62	short[4]	offm_a	slope term for offset calcs in microns

66	short	chksum_d	checksum
67	short[3]	fill_1	spares
		measprep data	
70	unsigned short	mask	type of measurement mask
71	unsigned short	calid	index of calibration data
72	unsigned short	timdel	vernier delay (packed)
73	unsigned short[2]	start_turns_skip_a	turns to skip before measuring
		regular measprep	
75	unsigned short	nthturn	how often to get data if multiple turns
76	unsigned short	navg	how many turns to average
		sin measprep	
75	unsigned short	nthturn	how often to get data if multiple turns
76	unsigned short	ndata	number of data to take
77	unsigned short	freq	frequency of sine wave in Hz
		scan measprep	
75	unsigned short	nthturn	how often to get data if multiple turns
76	unsigned short	ndata	number of data to take
77	unsigned short	navg	how many turns to average
		calibration measprep	
75	unsigned short	mask_flags	low nibble for attn; mux; multi-bunch
		abort measprep	
75	unsigned short	nthturn	how often to get data if multiple turns
78	short	chksum_d	checksum delta
79	short	fill_1	spare
		calibration results	
470	short[16]	ped_ampl	pedestals and amplitudes
486	short[4]	quad_err	quadrature angle errors in milliradians
490	unsigned short	mask_flags	low nibble for attn; mux; multi-bunch
491	unsigned short[2]	stat_a	status – set the 1 bit if good
493	short	chksum_d	checksum
494	short[4]	fill_1	spares
		nonaveraged results	
1030	unsigned short[4]	raw_a	semiraw amplitudes from I/Q channels combined after ped subtraction
		averaged results - 1	
1030	short	xerr	error on x mean (microns/10)
1031	short	yerr	error on y mean (microns/10)
1032	unsigned short	terr	error on tmit mean
1033	unsigned short[2]	goodmeas_a	number of good data in result
1035	unsigned short[4]	raw_a	semiraw amplitudes
		ave or nonave results	
1039	short	x	x result (2 micron units)
1040	short	y	y result (2 micron units)
1041	unsigned short	tmit	tmit scaled
1042	unsigned short	stat	status bit mask
		averaged results - 2	
1030	short[3]	data	x, y, and tmit
1033	unsigned short	stat	status bit mask
1034	short[3]	data_err	errors on mean of x, y, and tmit
1037	unsigned short[2]	goodmeas_a	number of good data in result
		timing results	
1030	unsigned short	tmit_1	average counts – A half
1031	unsigned short	tmit_2	average counts – B half
1032	unsigned short	stat	status bit mask

		sine results	
1030	short	x_phase	phase for A half in milliradians
1031	short	x_ampl	amplitude for A half in microns
1032	short	y_phase	phase for B half in milliradians
1033	short	y_ampl	amplitude for B half in microns
1034	short	x_rms	rms fit error for A half
1035	short	y_rms	rms fit error for B half
1036	unsigned short	stat	status bit mask
1043	short[5]	fill_1	spares
		scan results -1	
1048	short	x	x result
1049	short	y	y result
1050	unsigned short	tmit	beam intensity
		scan results -2	
1048	short[3]	data	x, y, and tmit
4120	short[2420]	fill_2	spares
		beam abort results	
6540	unsigned short	mask	mask with flags
6541	unsigned short	ndata	number of pulses
6542	unsigned short	last_good_index	index of last good measurement
6543	short[8400]	abort_data	x, y, and tmit
14943	short[1326]	fill_3	spares
		cal refine params	
16269	short[8]	ped_delta	pedestal correction
16277	short[4]	qi_ratio_delta	Q/I ratio correction
16281	short[4]	del_quaderr_delta	quadrature angle error correction
16285	unsigned short	ref_stat	status of refine calibration
16286	short[8]	cal_offset	calibration gain sine offset
		cal status checks	
16294	unsigned short	ped_stdev	max ped standard deviation
16295	unsigned short	ampl_dev	max diff amplitude vs expected
16296	unsigned short	ampl_rms	max rms fit error
16297	short[4]	dum_cal_phase	(placeholder)
16301	unsigned short[8]	orig_ampl_a	ampl before refinement step
16309	unsigned short[8]	xtra_ampl_a	ampl after refinement step
16317	short	fill_1	spare
		reserved	
16318	short	reserved	interrupt
16319	unsigned short	prep_id	measprep id for yy

6.2.2. Programming examples

The following examples show the steps in various operations of the RInQ module from initialization through obtaining results.

- Write the values on the crate_init structure.
- To calibrate: write the values on calibration measprep, set measprep 0.
- To measure closed orbit: write the values on the averaging measprep, set measprep 0.
- To measure turn-by-turn: write the values on the scan measprep, set measprep 0.

- Read results (calibration, measurement, or raw).

6.2.2.1. Initialization values

HSTA_BPMS_A(00H,00H):	00041H	x-only BPMS
HSTA_BPMS_A(01H,00H):	00000H	
HSTA_BPMS_A(00H,01H):	00081H	y-only BPMS
HSTA_BPMS_A(01H,01H):	00000H	
HSTA_BPMS_A(00H,02H):	00041H	
HSTA_BPMS_A(01H,02H):	00000H	
HSTA_BPMS_A(00H,03H):	00081H	
HSTA_BPMS_A(01H,03H):	00000H	
HSTA_BPMP_A(00H):	00001H	
HSTA_BPMP_A(01H):	00001H	
PCMM_A(00H,00H):	00000H	
PCMM_A(01H,00H):	00000H	
PCMM_A(02H,00H):	00000H	
PCMM_A(03H,00H):	00000H	
PCMM_A(04H,00H):	00000H	
PCMM_A(05H,00H):	00000H	
PCMM_A(00H,01H):	00000H	
PCMM_A(01H,01H):	00000H	
PCMM_A(02H,01H):	00000H	
PCMM_A(03H,01H):	00000H	
PCMM_A(04H,01H):	00000H	
PCMM_A(05H,01H):	00000H	
TCNST_A(00H):	0051EH	
TCNST_A(01H):	0051EH	
SINGLE_BUNCH_TMIT_A(00H):	00007H	
SINGLE_BUNCH_TMIT_A(01H):	00007H	
SPARE:	00000H	
ANGL_A(00H):	00000H	
ANGL_A(01H):	00000H	
OFFS_A(00H):	00000H	
OFFS_A(01H):	00000H	
OFFS_A(02H):	00000H	
OFFS_A(03H):	00000H	
MYSPARE_A(00H):	00000H	
MYSPARE_A(01H):	00000H	
LO_FREQ_OFFSET_HZ:	00000H	
CP_FREQ_OFFSET_HZ:	00000H	
LO_FREQ_OFFSET:	00000H	952.000 MHz
CP_FREQ_OFFSET:	0000FH	952.015 MHz
DBG_FLAG:	00001H	
N_CAL_MULT:	00005H	5 x 133.6 kHz

DBG_NOCHK: 010E1H
CHKSUM_D: 0FB97H

6.2.2.2. Perform calibration

The following values are written for a calibration measprep (address 70):

00020H mask, for calibration request
00000H calibration index 0
00000H vernier time delay (packed)
00000H turns to skip before measuring (lower)
00000H turns to skip before measuring (upper)
01004H mask flags, atten=4, mux=0, multi-bunch
00000H spare
00000H spare
00199H checksum, not important if crate dbg_nochk set

At this point, measprep=0.

6.2.2.3. Read calibration results

Read the dual-port memory addresses 470 to 492.

6.2.2.4. Closed orbit measurement

The following values are written for an averaging measprep (address 70):

00001H mask, for single measurement with averaging
00000H use calibration index 0
00000H vernier time delay (packed)
00000H turns to skip before measuring (lower)
00000H turns to skip before measuring (upper)
00001H measure every turn, nthturn=1
00400H average 1024 measurements, navg=1024
00000H spare
00199H checksum, not important if crate dbg_nochk set

At this point, measprep=0.

6.2.2.5. Read closed orbit results

Read the dual-port memory addresses 1030 to 1042.

6.2.2.6. Turn-by-turn measurement

The following values are written for a scan measurement (address 70):

00001H	mask, for scan data (array of results)
00000H	use calibration index 0
00000H	vernier time delay (packed)
00000H	turns to skip before measuring (lower)
00000H	turns to skip before measuring (upper)
00001H	measure every turn, nthturn=1
00400H	take 1024 measurements, ndata=1024
00001H	navg=1
00199H	checksum, not important if crate dbg_nochk set

At this point, measprep=0

6.2.2.7. Read turn-by-turn results

Read the dual-port memory addresses 1048 to 4119; format: x (microns), y (microns), tmit.

6.2.3. Include file

```
/*=========================================================================

  Abs:  Structures to be used for PEPII BPM software and firmware.
        These are to be used by VAX, micro and DSP software.
        Note we are purposely trying not to be dependant on
        other control system include files whose file names may be
        more than 8 characters, because DSP development is on the PC.

        Note that all items in structures on the dual-port memory of the DSP
        must fit within 16 bits, since the memory is only 16-bits wide.
        But also note that the DSP C compiler will think "short"s are
        32 bits so careful handling is needed when the DSP reads signed
        shorts from the dual-port memory.

  Name: pb_struc.hc

  Prev: none

  Auth: 25-Jul-1995, Linda Hendrickson (LJH)
  Rev:  04-Mar-1996, Alan Cheilek (CHEILEK)

************************          IMPORTANT          ************************
** Don't change the size of anything in here without taking great care!
** The size of the pb_dual_ts structure must not change!!!
** If you change calibration structures, please change SLCTXT:PB_STRUC.TXT
************************          IMPORTANT          ************************
-------------------------------------------------------------------------
  Mod:
        17-Nov-2000, Karey Krauter (KEK)
          Spare out pb_summary_ts.analog_attn which was only used for
          debugging and now we've changed it so it can't be just one field
          now, so it's easier to just spare it out.
```

```
        09-Dec-1999, Linda Hendrickson (LJH) and KEK
          Add offm array to pb_crate_ts. Incompatible change
          because we have to move the chksum_d.
        02-Sep-1999, Linda Hendrickson (LJH):
          Save orig_ampl_a in scratch area for Ron Johnson.
        30-Jul-1999, Linda Hendrickson (LJH):
          In pb_meas_xtra_u, change raw to raw_a[4].
        12-Nov-1998, Linda Hendrickson (LJH):
          Add items for new refine calibration mode.
        30-Oct-1998, Linda Hendrickson (LJH):
          Add comments and new define for angl scaling, to clarify it.
        23-Jul-1998, Linda Hendrickson (LJH):
          Add defines for separate PCMM scaling for 3rd order terms.
        19-Jul-1998, Linda Hendrickson (LJH):
          Add last_cal_id to summary_ts.
        21-Jan-1998, Linda Hendrickson (LJH):
          Add raw data output for single measurement.
        11-Jun-1997, Linda Hendrickson (LJH):
          Add skip first n turns option to measprep.
        10-Feb-1997, Linda Hendrickson (LJH):
          Create scratch structure; and put some scratch values there.
          Change summary structure per Brooks.
        06-Jan-1997, Linda Hendrickson (LJH):
          Remove timdel from crate-init and put it in all measpreps.
        19-Jun-1996, Linda Hendrickson (LJH):
          ADC data is bipolar; buffer uncompressed data,
          so I can't save as many pulses.

==========================================================================*/

/*
** The following key value is used in crate_ps->dbg_nochk.
*/

#ifndef PB_STRUC_HC              /* guard macro */
#define PB_STRUC_HC              /* guard macro */
#define PB_DBG_NOCHK 4321        /* key value to get around checksum */

/*
** The following key value is used for a measprep id to initialize
** the dual-port memory with the addresses equal to the value.
** Note if this happens, need to reinitialize the values in the
** dual-port memory to get the module to function.
*/

#define PB_MEASPREP_INIT 4321

/*
** The following mask bits apply to crate_ps->dbg_flag.
*/

#define PB_DBG_ALTCLOCK 1       /* use alternate clock speed, not 29.75 MHz */
#define PB_DBG_MEAS 2           /* use calib pulser for meas data */
#define PB_DBG_BUFFRAW 4        /* buffer raw results for cal, meas, sine */
#define PB_DBG_FAKECAL 8        /* make fake calibration data (not supported) */

#define PB_MAX_MEASPREP 40      /* 40 different measpreps are supported */
```

```
#define PB_MAX_CALIBS 20        /* max number of calibrations supported */
#define PB_MAX_SCANSTEPS 1024   /* max number of scan steps supported */

      /* Note that the software will support up to PB_MAX_ABORTDATA
         scan steps but for more than 1024, it uses the abort area. */

#define PB_MAX_SINSTEPS 1024    /* max number of steps supported for sine meas */
#define PB_SIZE_DUAL 16384      /* size of dual-port memory */
#define PB_TOTBPMS 4            /* total number of BPMS possible (HER/LER X/Y) */
#define PB_TOTMUXES 2           /* 2 possible mux settings for HER/LER */
#define PB_BPMS_PER_MUX 2       /* max 2 BPMS per mux step */
#define PB_TOT_HSTA_WORDS 2     /* 2 16-bit words in a BPMS or BPMP HSTA */
#define NUM_PB_ADC_CHANS 8      /* 8 channels of ADC data */
#define NUM_PB_ANGLS 4          /* 4 quadrature angle errors */
#define PB_MAX_ABORTDATA 2800   /* size of abort ring buffer */
#define PB_N_RESULTS 3          /* how many data items is x,y,tmit */
#define PCMM_WORDS 6            /* 6 PCMM values per BPM */
#define PCMM_LOWORDER_WORDS 2   /* 1st 2 PCMMs are first order; rest are 3rd*/

#define PB_MAX_VDEL 256         /* 8 bits is max vernier delay */

/*
** Sizes of spares in pb_dual_ts.
*/

#define PB_SPARE1 5             /* 1st spares */
#define PB_SPARE2 2420          /* 2nd spares */
#define PB_SPARE3 1326          /* 3rd spares */

#define PB_EXTRA_RAW 64         /* extra pieces of raw data */

/*
** Camac addresses for dual-port memory, in 16-bit words: For micro code,
** the following MUST correspond to the order in pb_dual_ts.
*/

#define PB_SUMMARY_ADDR 0
#define PB_CRATE_ADDR (PB_SUMMARY_ADDR + (sizeof(pb_dsp_summary_ts)/2))
#define PB_MEASPREP_ADDR (PB_CRATE_ADDR + (sizeof(pb_dsp_crate_ts)/2))
#define PB_CAL_ADDR (PB_MEASPREP_ADDR +
                  (sizeof(pb_dsp_measprep_ts)*PB_MAX_MEASPREP/2))
#define PB_MEAS_ADDR (PB_CAL_ADDR + (sizeof(pb_dsp_cal_ts)*PB_MAX_CALIBS/2))
#define PB_MEASSCAN_ADDR (PB_MEAS_ADDR + (sizeof(pb_dsp_meas_tu)/2)+ PB_SPARE1)
#define PB_ABORT_ADDR (PB_MEASSCAN_ADDR +
                  (sizeof(pb_dsp_measscan_ts)*PB_MAX_SCANSTEPS/2)+ PB_SPARE2)
#define PB_BIGSCAN_ADDR (PB_ABORT_ADDR + (sizeof(pb_abort_header_ts)/2))

/*
** The following bits are for status in pb_summary_ts:
*/

#define PB_DSPSTATUS_GOOD     (0x0001) /* successful */
#define PB_DSPSTATUS_BADMEM   (0x0014) /* memory checks were unsuccessful */
#define PB_DSPSTATUS_BADSTRUC (0x0104) /* Dual-port memory structure wrong
                                          size; code error! */
```

```
/*
** The following bits are for meas_status in pb_summary_ts:
*/

/*      PB_DSPSTATUS_GOOD      (0x0001), successful */

#define PB_MEASSTAT_INCOMPLETE (0x0004) /* Measurement did not complete.
        If only this bit is set, probably no triggers or YY too soon. */
#define PB_MEASSTAT_BADCRATE   (0x0024) /* failed crate init checksum */
#define PB_MEASSTAT_BADCAL     (0x0044) /* failed calibration checksum */
#define PB_MEASSTAT_BADMEASP   (0x0084) /* failed measprep checksum */
#define PB_MEASSTAT_DSPSLOW    (0x0104) /* DSP too slow: missing some ADC data */
#define PB_MEASSTAT_ADCSLOW    (0x0204) /* ADC timed out, missing triggers? */
#define PB_MEASSTAT_BADID      (0x0404) /* invalid measprep id */
#define PB_MEASSTAT_BADCALID   (0x0804) /* invalid calibration id */
#define PB_MEASSTAT_DBERR      (0x1004) /* invalid database (crate-init) values*/
#define PB_MEASSTAT_BADVDEL    (0x2004) /* invalid vernier delay requested */
#define PB_MEASSTAT_BADPTS     (0x8004) /* too many points requested */

/*
** The following bits are for mask in pb_measprep_ts:
*/

#define PB_MEASP_MEAS           (0x0001) /* regular measurement or averaging*/
#define PB_MEASP_TIM            (0x0002) /* timing diagnostic */
#define PB_MEASP_SIN            (0x0004) /* sine fit */
#define PB_MEASP_SCAN           (0x0008) /* scan data */
#define PB_MEASP_CAL            (0x0020) /* calibration */
#define PB_MEASP_ABORT          (0x0040) /* beam abort */
#define PB_MEASP_REFINECAL      (0x0080) /* refine calibration */

/*
** Note flipping x or y will probably only happen on the micro, but
** it's supported on the DSP with the following.
** Note especially that if and X-only and a Y-only BPM are in the
** "wrong" order i.e. Y first, these REALLY refer to first and
** 2nd only BPM.
*/

#define PB_MEASP_FLIPX          (0x0100) /* flip x (or first y if 2 Y-only) */
#define PB_MEASP_FLIPY          (0x0200) /* flip y (or second x if 2 X-only) */

/*
** Note the following bit is for very slow measurement.  When it
** is set, if there are multiple measurements to take for averaging or
** for a scan, there is such a long time between measurements that
** the real-time code times out.  This might be used, for example,
** if we want to do a scan on injected beam at 120 Hz. Only for regular
** measurements; not supported for beam abort or calibration.
*/

#define PB_MEASP_SLOW      (0x0400) /* very slow measurement */

/*
** The following are associated with mask_flags in pb_vax_cal_ts:
*/
```

```
#define PB_2ND_MUX          (0x100) /* set if 2nd mux setting */
#define PB_MULTIBUNCH      (0x1000) /* set if multibunch calibration */

/*
** The following are for mask in pb_dsp_abort_ts:
*/

#define PB_ABORT_FILLED    (0x0001) /* set if the abort buffer fills up */
#define PB_ABORT_ACQUIRING (0x0002) /* set if acquiring data now */
#define PB_ABORT_CALCING   (0x0004) /* set if calculating now */
#define PB_ABORT_DONE      (0x0008) /* set if successfully done with results*/
#define PB_ABORT_REQUESTED (0x0010) /* set if abort request is pending */

/*
** The following constants are for scaling results to 16 bits:
*/

#define PB_MAX_PCMM 50            /* for scaling PCMM 1st order coeffs */
#define PB_MAX_PCMM_3RD 0.1       /* for scaling PCMM 3rd order coeffs */
#define PB_MAX_TCNST 50           /* for scaling TCNST */
#define PB_MAX_SHORT 65536        /* scale to 16 bits */
#define PB_MAX_SSHORT 32768       /* scale to signed short 15 bits */
#define PB_CONVERT_ANGL_DB 0.1    /* multiplier of short to get ANGL, in degrees
         (For micro code:  angles in database are degrees, but we pass them in
         degrees/10, so divide by this to go from the database to the firmware.) */
#define PB_CONVERT_ANGL 0.0017453 /* multiplier of short to get ANGL, in rad
         (This is for the firmware only: from deg/10, this is 2*PI/360/10.)  */
#define PB_CONVERT_OFFS 1.0E-3    /* multiplier of short to get OFFS, in mm */
#define PB_MAX_ADC 8192           /* max value of 14-bit ADC if bipolar */
#define PB_CONVERT_QUADERR 1.0E-3 /* multiplier of short to get quad err,in
                                     radians*/
#define PB_CONVERT_XY 500         /* multiplier of x&y (in mm) to get short
                                     2-microns*/
#define PB_CONVERT_MICRONS 1000   /* multiplier (in mm) to get a short microns */
#define PB_CONVERT_XYERR 10000    /* multiplier of x&y errs (in mm) to get short*/
#define PB_MAX_TMIT 6.0E11        /* maximum tmit for scaling; 29Jun99;was 1.e11 */
#define PB_CONVERT_TMIT (65536/PB_MAX_TMIT)  /* divisor of short to get tmit */

/*
** The following are for checks and scaling of calibration refinement.
*/

/* #define PB_MAX_PEDDELTA 10 */    /* max ped change, counts */
#define PB_MAX_PEDDELTA 40          /* max ped change, counts */
#define PB_MAX_QIDELTA .1           /* max Q/I ratio change, ratio */
/* #define PB_MAX_QUADERRDELTA 52 *//* max quad error change,milliradians, 3 deg */
#define PB_MAX_QUADERRDELTA  104    /* max quad error change,milliradians, 3 deg */
#define PB_MAX_PHIDELTA .02777      /* max phi change between turns, periods;
                                       10  deg*/

/*
** The following status bits are used for ref_stat, in scratch_ts,
** for refine calibration.
*/

#define REF_BAD_PEDDELTA      (0x0010)  /* ped delta too large */
#define REF_BAD_QIDELTA       (0x0020)  /* qi ratio delta too large */
```

```
#define REF_BAD_QUERRDELTA    (0x0040)  /* quadrature angle delta too large */
#define REF_BAD_PHIDELTA      (0x0080)  /* bad phi delta in meas */
#define REF_BAD_SAT           (0x0100)  /* saturated */
#define REF_BAD_SUMLO         (0x0200)  /* sum low; no beam */

/*
** The following bits are used for status for x and y.
** The status goes into the second-highest bit of the 16-bit x or
** y data. The highest bit is the sign bit.
*/

#define PB_BAD_DATABIT      0x4000      /* set this bit if it's bad */
#define PB_SIGN_BIT         0x8000      /* sign bit */
#define PB_MAX_XY           32.767      /* In mm 14 bits is max x or y in 2
                                           microns, because we need stat and
                                           sign bits 15 and 16. This is
                                           checked on the DSP. */

/*
** The following are for gains for phase calcuation.  (0 to disable).
** These are in the crate-init structure.
*/

#define PB_GAIN_CONVERT 0.01 /* phase gains on dual-port are in percent,
                                multiply by this to get gains to use */
#define PB_MIN_GAIN 0        /* minimum gain factor, % */
#define PB_MAX_GAIN 100      /* max gain factor, % */

/*
** Note:  All of the checksums between the micro and dsp
**        have the following definition:
**    Add up all the values in the memory area including chksum_d.
**    The sum must be equal to PB_CHKSUM_CHECK (in PB_CHECK.C).
**    So chksum_d isn't the checksum, it's the quantity needed to
**    add up to the right value.  This way, if we add items later
**    to these structures, we can add them after the chksum_d and
**    everything should work fine, so it's semi-forward-compatible.
*/


/*
*************************************************************************
******************          Summary Data Structures          ******************
*/

typedef struct
{
   unsigned short version_number;  /* version number of running firmware */
            /* this is nn in the filename PEPBPM:PBMAIN.EXT;nn */
   short revision_number;          /* module revision number first 8 bits */
            /* 16 bit is production or dev: set if dev.
               If negative, it's the development module. */
   short serial_number;            /* serial number of this module */
            /* Module serial number in first 10 bits.
               Module type bits 11-16: (RInQ type is 7). */
   unsigned short last_cal_id;     /* cal index of last cal used */
```

```
   unsigned short status;          /* status of the DSP boot and init */
   unsigned short meas_status;     /* status of the last measurement */
   unsigned short last_prep_id;    /* index of last measprep recieved */
   unsigned short last_iof;        /* copy of iof register */

/*
** The following are from pb_inith, for debugging:
*/

   unsigned short lo_freq[2];     /* lower and upper bits of lo freq word */
   unsigned short spare;          /* used to be analog attenuation for debugging */
   unsigned short analog_ctrl;    /* analog control: muxing */
   unsigned short timdel;         /* timing delay register, 8 bits */

/*
** The following are from pb_initg, for debugging calibration:
*/

   unsigned short cp_freq[2];      /* lower and upper bits of cp freq word */
   unsigned short cal_analog_ctrl; /* analog control: for calib */

} pb_summary_ts;         /* structure with summary data, for micro,dsp */

/*
** The following structure is used between the micro and DSP software
** for summary data, plus spares.
*/

typedef struct
{
  pb_summary_ts pb_summary_s; /* holds the summary data */
   short fill_1[2];           /* spares */
} pb_dsp_summary_ts;      /* structure with summary data+spares, for micro,dsp */


/*
*******************          Crate Init Structures           *******************
*/

typedef struct
{
   unsigned short HSTA_BPMS_a[PB_TOTBPMS][PB_TOT_HSTA_WORDS];
        /* HSTA for each of 4 possible BPMs. Just put in a zero
           if the BPMS is not needed.  There are 2 16-bit words in
           the HSTA: lower bits are first, then the upper bits. */
   unsigned short hsta_bpmp_a[PB_TOT_HSTA_WORDS];
        /* BPMP HSTA; lower then upper bits */

/*
** The order of the BPMS in these arrays of 4 is:
**      - First (or X-only) BPMS for the first mux setting.
**      - Second (or Y-only) BPMS for the first mux setting (put zeros
**                in here if a second BPMS isn't used).
**      - First (or X-only) BPMS for the second mux setting.
**      - Second (or Y-only) BPMS for the second mux setting (put zeros
**                in here if a second BPMS isn't used).
*/
```

```
/*
** Note that the 1st 2 PCMM values and all tcnst are scaled as follows:
**      Since their values are always positive and never more than
**      50.0, we scale the floating-point values into the shorts with:
**            short_value = float_value*(2**16)/50.
**      The scaling for the last 4 PCMM values (3rd order corrections) is
**             short_value = float_value*(2**15)/0.1.
**            (These can be negative and are small values.)
** Note that we can have up to 6 PCMM values for each of 2 mux
**      settings. In order they are from SSMITH's latest spec
**      "Position and Beam Current Algorithm":
**      PCMMx, PCMMy, a11, a12, a21, a22  for an XY BPM.
**      If we have an X-only and a Y-only BPM, we use
**      PCMMx, PCMMy, a12, a22, spare, spare.
** Note the a11...a22 terms are not used unless the BPMS HSTA
**      bit HSTA_3RDORD is set.
*/

   unsigned short pcmm_a[PB_TOTMUXES][PCMM_WORDS];      /* PCMM values */

/*
** Note the tmit multipliers tcnst and tmit_mult are for the first
** BPM, since you only get one tmit. So there's only one per mux setting.
*/

   unsigned short tcnst_a[PB_TOTMUXES];     /* tmit multipliers: SNRM*TNRM */
   unsigned short single_bunch_tmit_a[PB_TOTMUXES];
/* Correct tmit for single-bunch vs multi-bunch, not scaled:
      single-bunch-tmit = multi-bunch-tmit/tmit_mult. See SSmith: This might be
      just 2 hardcoded numbers, each=7. */
   unsigned short sspare;       /* a spare word */
   short angl_a[PB_TOTMUXES];   /* Angles from horizontal to BPMS x-dir, in deg/10.
                                   It isn't valid unless just an XY BPM */
   short offs_a[PB_TOTBPMS];    /* offsets to subtract from x/y, in microns */
   unsigned short myspare_a[2]; /* spares */
   short lo_freq_offs_hz;       /* offs freq of l.o. addition, Hz */
   short cp_freq_offs_hz;       /* offs freq of cal addition, Hz */
   short lo_freq_offset;        /* offset frequency of local oscillator
                                   during calibration, KHz (probably -67) */
   short cp_freq_offset;        /* offset frequency of calibration pulser
                                   during calibration, KHz (probably 68) */
   unsigned short dbg_flag;     /* see PB_DBG masks above */
   short n_cal_mult;            /* multiplier of trig_freq to add to calibrator
                                   frequency*/
   short dbg_nochk;             /* if == PB_DBG_NOCHK, ignore chksum */

/*
** Note the following apply to phase corrections during real-time acquisition.
** Enter either = 0 to disable it. Phase calculations do not apply during
** calibration or timing, but they apply during all other types of acquisitions.
*/

   unsigned short phase_gain1; /* gain for 1st phase calc step, percent */
   unsigned short phase_gain2; /* gain for subsequent phase calc steps, % */
   short offm_a[PB_TOTBPMS];   /* slope terms for offsets per Ron Johnson
                                  in microns */
```

```
   short chksum_d;             /* checksum delta */
} pb_crate_ts;                 /* structure with crate init data, for micro,dsp */

/*
** The following structure is used between the micro and DSP software
** for crate-init data, plus spares.
*/

typedef struct
{
  pb_crate_ts pb_crate_s;      /* holds the crate-init data */
   short fill_1[3];            /* spares */
} pb_dsp_crate_ts;             /* structure with crate-init data+spares, for
                                  micro,dsp */



/*
********************          Measprep Structures          **********************
*/

typedef struct
{
   unsigned short nthturn; /* how often to get data if multiple-turns */
   unsigned short navg;    /* number of pulses to average */
} pb_measm_ts;             /* measprep structure for regular measurement */

typedef struct
{
   unsigned short nthturn; /* how often to get data if multiple-turns */
   unsigned short ndata;   /* number of data to take */
   unsigned short freq;    /* frequency of sine wave, Hz */
} pb_meassin_ts;           /* measprep structure for sine measurement */

typedef struct
{
   unsigned short nthturn; /* how often to get data */
   unsigned short ndata;   /* number of data to take */
   unsigned short navg;    /* number of pulses to average within each data*/
} pb_measscan_ts;          /* measprep structure for scan measurement */

typedef struct
{
   unsigned short mask_flags;  /* attenuation, mux setting, multi-bunch flag */

/*
** The 2 bytes of the mask_flags are arranged as follows:
**    000F    lowest nibble is for the first half (low half) attn
**    00F0    second nibble is for the second half (high half) attn
**    0100    mux setting.  See defaults.dbs MUXSHER and MUXSLER.
**    1000    multi-bunch processing.
*/

} pb_measCal_ts;           /* measprep structure for calibration request */

typedef struct
{
```

```
   unsigned short nthturn; /* how often to get data */
} pb_measabort_ts;         /* measprep structure for beam abort measurement */

typedef struct
{
   unsigned short mask;    /* mask indicating type of measurement */
   unsigned short calid;   /* Index of calibration to use, needed for all including
                              timing. For calibration, results are put here. */
   unsigned short timdel;  /* vernier delay for x and y, packed for hardware, 8
                              bits for each */
   unsigned short start_turns_skip_a[2]; /* turns to skip before measuring,
                                            lower then upper bits of 32 bit
                                            unsigned word */
   union
   {
     pb_measm_ts pb_measm;         /* regular measurement or averaging */
     pb_meassin_ts pb_meassin;     /* sine fit of scanned data */
     pb_measscan_ts pb_measscan;   /* scan (or buffered) data */
     pb_meascal_ts pb_meascal;     /* performing a calibration */
     pb_measabort_ts pb_measabort; /* taking beam abort data */
      /* Note the refine calibration has no addition data required. */
   } pb_measprep_u;      /* union of measprep data for various types */
   short chksum_d;       /* checksum delta */
} pb_measprep_ts;        /* structure with measprep data, for micro,dsp */

/*
** The following structure is used between the micro and DSP software
** for measprep data, plus spares.
*/

typedef struct
{
  pb_measprep_ts pb_measprep_s;  /* holds the measprep data */
   short fill_1;                 /* spare */
} pb_dsp_measprep_ts;            /* structure with measprep data+spares, for
                                    micro,dsp */


/*
******************          Calibration Structures          ******************
*/

typedef struct
{
    short pedestal;
    short ampl;

/*
** These are bipolar readouts from a 14-bit ADC but are averaged.
** They are passed around scaled to 16 bits.  So to convert them back
** to counts for the display and for calculations, for example:
**            ampl_float = ampl*(2**14)/(2**16).
*/

} pb_ped_ampl_ts;     /* structure to hold calibration data for 1 ADC channel */

/*
```

```
** The following structure is used by the VAX, micro and DSP software
** to pass around calibration data.
*/

typedef struct
{

/*
** Note a BPM has 4 channels (call them CH1-CH4), with I and Q
** for each BPM channel, so there are a total of 8 ADC channels.
** Normally, the order of the BPM channels is +x, -x, +y, -y.
*/

    pb_ped_ampl_ts ped_ampl_as[NUM_PB_ADC_CHANS]; /* ped,ampl */

/*
** For each ADC channel in the following order:
** CH1 I, CH1 Q, CH2 I, CH2 Q, CH3 I, CH3 Q, CH4 I, CH4 Q
*/

    short quad_err_a[NUM_PB_ANGLS];  /* quadrature angle errors between I,Q */

/*
** Errors between I,Q for BPM channels 1-4, in integral milli-radians.
*/

    unsigned short mask_flags;  /* attenuation, mux setting, multi-bunch flag */

/*
** The 2 bytes of the mask_flags are arranged as follows:
**    000F    lowest nibble is for the first half (low half) attn
**    00F0    second nibble is for the second half (high half) attn
**    0100    mux setting
**    1000    multi-bunch processing.
*/

    unsigned short stat_a[PB_BPMS_PER_MUX];   /* status bits */

/* Note that the first stat is for the lower half of the BPM.
For X or Y-only BPMS, it goes with the BPMS of the lower unit
number. The second word is for the higher X or Y-only unit number.
For an XY BPMS, both words are set to the same value, so looking
at the first one is sufficient. */

} pb_vax_cal_ts;    /* structure with calibration data for vax,micro,dsp */

/*
** The following structure is used between the micro and DSP software
** for calibration data. Note this structure is not for use by the VAX.
*/

typedef struct
{
  pb_vax_cal_ts pb_vax_cal_s;   /* holds the calibration data */
   short chksum_d;              /* checksum of pb_cal_s data */
} pb_cal_ts;         /* structure with calibration data, for micro,dsp */
```

```
/*
** The following structure is used between the micro and DSP software
** for calibration data, plus spares.  Note this structure is not
** for use by the VAX.
*/

typedef struct
{
  pb_cal_ts pb_cal_s;           /* holds the calibration data */
   short fill_1[4];             /* spares */
} pb_dsp_cal_ts;  /* structure with calibration data+spares, for micro,dsp */


/*
****************         Measurement Results Structures         ****************
*/

/*
** Note in all measurement results, x and y are in 2-micron units.
** x and y errors are in microns/10.
** TMIT and TERR are between 0 and 1.E11:
** TMIT_short = (TMIT_float)*(2**16)/1.E11.
*/

typedef struct           /* this structure is for nonaveraged meas data */
{
   unsigned short raw_a[NUM_PB_ANGLS]; /* semiraw amplitude from I/Q chans,
                                          combined after ped subtraction */
} pb_measnoavg_ts;      /* measurement results not averaged*/

typedef struct           /* this structure is for averaged meas data */
{
   short xerr;           /* error on x mean, (microns/10)  */
   short yerr;           /* error on y mean, (microns/10) */
   unsigned short terr;  /* error on tmit mean */
   unsigned short goodmeas_a[PB_BPMS_PER_MUX];
        /* Number of good data included in the result */
        /* for x and for y. (If an XY BPMS, they're the same.) */
   unsigned short raw_a[NUM_PB_ANGLS]; /* semiraw amplitude from I/Q chans,
                                          combined after ped subtraction */
} pb_measavg_ts;         /* measurement results for averaged */

typedef union
{
  pb_measnoavg_ts measnoavg_s;  /* nonaveraged single shot data */
  pb_measavg_ts measavg_s;      /* averaged single shot data */
} pb_meas_xtra_tu;  /* union for extra measurement results */

typedef struct      /* this structure is for single shot or avgd meas data */
{
   short x; /* x result, 2-microns: value or mean */
            /* (or, if there are 2 Y-only BPMs: y for the first BPM) */
            /* Note bits PB_BAD_DATABIT and PB_SIGN_BIT apply to x and y */
   short y; /* y result, 2-microns: value or mean */
            /* (or, if there are 2 X-only BPMs: x for the second BPM) */
            /* Note bits PB_BAD_DATABIT and PB_SIGN_BIT apply to x and y */
```

```
   unsigned short tmit;  /* beam intensity:  value or mean */
   unsigned short stat;  /* status bit mask */
/* Note that if the stat is bad, this apply to either x or y or both.  Need to look
at the PB_BAD_DATABIT in x and y to know if the bad stat applies. */
  pb_meas_xtra_tu pb_meas_xtra_u; /* extra meas results for averaged or non */
} pb_meas_ts;            /* measurement results */

typedef struct           /* this stucture is for averaged meas data */
{
   short data[PB_N_RESULTS];     /* x, y and tmit */
   unsigned short stat;          /* status bit mask */
   short data_err[PB_N_RESULTS]; /* errors on mean of x, y and tmit */
   unsigned short goodmeas_a[PB_BPMS_PER_MUX];
             /* number of good data included in the result for x and for y.
                (If an XY BPMS, they're the same.) */
} pb_meas_array_ts;       /* another form of pb_meas_ts measurement results */

typedef struct
{

/*
** These tmits are average of the quadrature sum of the I and Q ADC counts.
** So these are numbers that fit into 13 bits, and they're returned scaled
** to 16 bits. (So they're amplitude * 8, or PB_MAX_SHORT/PB_MAX_ADC).
*/

   unsigned short tmit_1;   /* avg counts for an XY BPM, or the X-only BPM,
                               (or the first Y-only if 2 Y-only BPMs) */
   unsigned short tmit_2;   /* tmit for the Y-only BPM (or 2nd X-only BPM),
                                 if there is one */
   unsigned short stat;     /* status bit mask */
/* Note the stat is always GOOD for timing unless we failed to take data; so we
don't need to differentiate if x or y is bad.  */
} pb_tim_ts;                /* timing results */
typedef struct
{

/*
** Note in this structure, scaling is:
**    Phases are in integral milliradians, like quad angle error
**    Amplitudes are scaled in microns.
**    RMS fit errors are in microns also.
*/

   short x_phase;       /* phase for x (or first Y-only), milliradians */
   short x_ampl;        /* amplitude for x (or first Y-only), microns */
   short y_phase;       /* phase for y  (or 2nd X-only) */
   short y_ampl;        /* amplitude for y  (or 2nd X-only)  */
   short x_rms;         /* rms fit error for x (or 1st Y-only), microns */
   short y_rms;         /* rms fit error for y  (or 2nd X-only), microns */
   unsigned short stat; /* status bit mask */
                        /* If the x or 1st Y-only fit is bad, use STAT_XBAD
                           If the y or 2nd X-only fit is bad, use STAT_YBAD. */
   unsigned short goodmeas_a[PB_BPMS_PER_MUX];
               /* number of good data included in the result for x and for y.
                  (if an XY BPMS, they're the same */
} pb_sin_ts;    /* sine results */
```

```
typedef union
{
  pb_meas_ts pb_meas_s;              /* measurement results */
  pb_meas_array_ts pb_meas_array_s;  /* another form of meas results */
  pb_tim_ts pb_tim_s;                /* timing results */
  pb_sin_ts pb_sin_s;                /* sine results */
} pb_dsp_meas_tu;                    /* union for measurement, timing or sine
                                        results */


/*
*******************          Scan Results Structure          *******************
*/

typedef struct
{
   short x;  /* x result */
             /* (or, if there are 2 Y-only BPMs: y for the first BPM) */
             /* Note bits PB_BAD_DATABIT and PB_SIGN_BIT apply to x and y. */
   short y;  /* y result */
             /* (or, if there are 2 X-only BPMs: x for the second BPM) */
             /* Note bits PB_BAD_DATABIT and PB_SIGN_BIT apply to x and y. */
   unsigned short tmit;               /* beam intensity */
} pb_dsp_measscan_ts;                 /* scan results structure */

typedef struct
{
   short data[PB_N_RESULTS];          /* x, y and tmit */
} pb_dsp_measscan_array_ts;           /* another form of scan results structure */


/*
*************       Beam Abort Measurement Results Structure       *************
*/

typedef struct
{
   unsigned short mask;                /* mask with flags */
   unsigned short ndata;               /* number of pulses, should be
                                          PB_MAX_ABORTDATA if filled */
   unsigned short last_good_index;     /* index of last good measurement */
} pb_abort_header_ts;

typedef struct
{
  pb_abort_header_ts header_s;        /* header of beam abort data */
  pb_dsp_measscan_ts pb_dsp_measscan_as[PB_MAX_ABORTDATA]; /* x,y,tmit */
} pb_dsp_abort_ts;                     /* scan results structure */

typedef struct
{

/*
** The following are for calibration refinement parameters.
** The calibration changes from the last refinement are stored here.
*/
```

```
  short ped_delta[NUM_PB_ADC_CHANS];     /* change to pedestal in cal refine */
   /* scale value by PB_MAX_PEDDELTA/PB_MAX_SSHORT to get counts */
  short qi_ratio_delta[NUM_PB_ANGLS];    /* change to q_ampl/i_ampl */
   /* scale value by 100*PB_MAX_QIDELTA/PB_MAX_SSHORT to get percent */
  short del_quaderr_delta[NUM_PB_ANGLS]; /* change to IQ quadrature angle err */
   /* scale value by PB_MAX_QUADERRDELTA/PB_MAX_SSHORT to get millradians */
  unsigned short ref_stat;               /* status for refine calib */

  short cal_offset[NUM_PB_ADC_CHANS];    /* calibration gain sine offset */

/*
** The following are results from status checks for the last calibration.
*/

   unsigned short ped_stdev;             /* max ped std deviation, counts */
   unsigned short ampl_del;              /* max difference ampl vs expected */
   unsigned short ampl_rms;              /* max rms fit error from sine */
   short dum_cal_phase[NUM_PB_ANGLS];    /* dummy placeholder */
   unsigned short orig_ampl_a[NUM_PB_ADC_CHANS]; /* ampl before extra step */
   unsigned short xtra_ampl_a[NUM_PB_ADC_CHANS]; /* amplit from extra step */
   short fill_1;                         /* spare */
} pb_scratch_ts;                         /* 49 words scratch structure */


/*
***************        Definition of the Dual-Port Memory        ***************
*/

/*
** The following structure defines how the dual-port memory is laid
** out. This memory is 16384 16-bit words.
*/

typedef struct
{
   pb_dsp_summary_ts pb_dsp_summary_s;
               /* 18 words, DSP-write: DSP summary data */
   pb_dsp_crate_ts pb_dsp_crate_s;
               /* 52 words, micro-write: crate-init data */
   pb_dsp_measprep_ts pb_dsp_measprep_as[PB_MAX_MEASPREP];
               /* 400 words, micro-write: meas-prep data */
   pb_dsp_cal_ts pb_dsp_cal_as[PB_MAX_CALIBS];
               /* 560 words, micro-write or DSP-write: calibration data */
   pb_dsp_meas_tu pb_dsp_meas_u;
               /* 13 words, DSP-write: results from single measurement */
   short fill_1[PB_SPARE1];
               /* 5 words: spares */
   pb_dsp_measscan_ts pb_dsp_measscan_as[PB_MAX_SCANSTEPS];
               /* 3072 words, DSP-write: results from scan measurement */
   short fill_2[PB_SPARE2];
               /* 2420 words: spares */
   pb_dsp_abort_ts pb_dsp_abort_s;
               /* 8403 words, DSP-write: beam abort data */
   short fill_3[PB_SPARE3];
               /* 1326 words: spares */
   short extra_raw[PB_EXTRA_RAW];
               /* buffer up some extra raw data */
```

```
   pb_scratch_ts pb_scratch_s;
                /* 49 words: scratch */
   short reserved;
                /* 1 word: reserved, interrupt */
   unsigned short prep_id;
                /* 1 word, micro-write: measprep id for yy, interrupts DSP. */

} pb_dual_ts;               /* structure which defines the whole dual-port memory */

#define PB_STRUC_DEF        /* indicates we've included this file */
#endif                      /* PB_STRUC_HC guard macro */
```

7. References

1. Shafer, R.E., Beam Position Monitoring, AIP Conf. Proc. **212,** 26-58 (1989).
2. IEEE Standard 1057, Digitizing Waveform Recorders (1989).
3. Aiello, G.R., A Digital Approach for Phase Measurement Applied to Delta-t Tuneup Procedure, Proc of the 1993 Particle Accelerator Conference, 2367-2369 (1993).
4. Aiello, G.R., Johnson, R.G., Martin, D.J., Mills, M.R., Olsen, J.J., and Smith, S.R., Beam Position Monitor System for PEP-II, AIP Conf. Proc. **390,** 341-349 (1996).
5. Johnson, R.G., Smith, S.R., and Aiello, G.R., Performance of the Beam Position Monitor System for the SLAC PEP-II B Factory, AIP Conf. Proc. **451,** 395-402 (1998).
6. Johnson, R.G. and Smith, S.R., Some Solved Problems with the SLAC PEP-II B Factory Beam-Position Monitor System, AIP Conf. Proc. **546,** 448-455 (2000).