SR1 Orbit Control Design Note

5.2.39.4 - Rev. 0

Date: 2008-03-28

Canadian Light Source Inc.
101 Perimeter Road
University of Saskatchewan
Saskatoon, Saskatchewan
S7N 0X4 Canada

	Signature	Date
	Original on File – Signed by:	
Author	D. Chabot	
Reviewer #1	R. Berg	
Reviewer #2	J. Vogt	
Approver	E. Matias	

BLANK PAGE

Revision History

Revision	*Date*	*Description*	*Author*
A	2008-03-25	Original Draft	Daron Chabot
0	2008-03-28	Issued for use.	Daron Chabot

BLANK PAGE

TABLE OF CONTENTS

BLANK PAGE

1.0 Introduction

This document examines structural and behavioral components of the CLS storage ring orbit control (SROC) system. The discussion is focused around the data acquisition aspects of that system. As part of an ongoing project, the existing SROC system has been replaced with a functionally equivalent system incorporating the real-time operating system, RTEMS. This decision was primarily motivated by a desire for improved quality positional data.

Functionally, the SROC system measures the particle beam position at various points around the storage ring (SR) and then applies trajectory adjustments based on those measurements by manipulating the fields of orbit correction magnets (OCM). In this way, the path of the particle beam may be controlled according to operational requirements, as found in [1] and [2].

Particle beam orbit quality is perturbed when it deviates from the magnetic axes of lattice elements. These perturbations arise due to element misalignment, vibration or thermal loading, and power supply fluctuations. The bandwidth of these orbit disturbances varies from DC to the low kilohertz range.

The brilliance of light produced by a synchrotron is inversely related to the emittance of the particle beam orbiting in the storage ring. Among other factors, the positional stability of the particle beam contributes to the maintenance of a small emittance, and therefore greater brilliance. Hence, the stability of the particle beam orbit is of significant operational importance.

1.1 ABBREVIATIONS

SR	Storage Ring
SROC	Storage Ring Orbit Control
BPM	Beam Position Monitor(s)
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
DIO	Digital Input/Output
OCM	Orbit Correction Magnet(s)
OCH	Orbit Correction magnet Horizontal
OCV	Orbit Correction magnet Vertical
EPICS	Experimental and Industrial Control System
IOC	Input/Output Controller
RTOS	Real-Time Operating System
RTEMS	Real-Time Executive for Multiprocessor Systems
DSP	Digital Signal Processing

2.0 SROC System Overview

The SROC system may be characterized according to the rate or bandwidth of the corrections applied to the beam trajectory. For the systems considered here, the rate of correction application is DC, or *static*: i.e. particle beam trajectory corrections are autonomously applied at a

rate of much less than 1 Hz (in fact, it is on the order of 0.1 Hz). In some cases, orbit revisions are also applied interactively by accelerator operators.

Physically, the SROC system is composed of the hardware and software items illustrated in the component-deployment diagram of Figure 1.

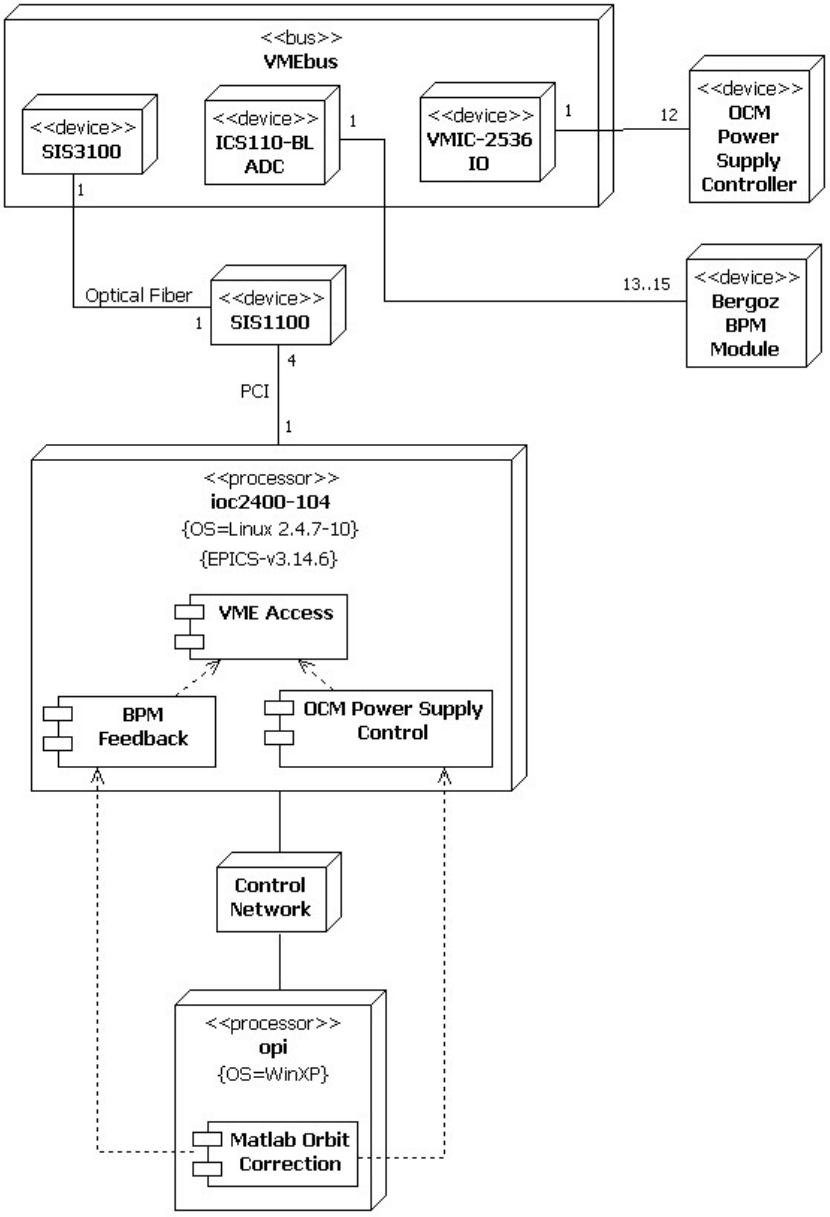

Figure 1 Component-deployment view of the SROC system prior to the implementation of the RTEMS-based system.

Key components of the SROC system are briefly outlined here:

1) **Beam position monitors (BPM)**: (not shown) symmetric configurations of 4 sensors, producing analog signals in proportion to the proximity of the beam passing through them.

2) **Bergoz BPM Modules**: sample BPM sets at 2.5 kHz, producing analog x-y coordinates of beam position. Given the 56 BPM sets in the storage ring, a total of 112 channels of analog positional data are produced at each sampling interval. These signals are then digitized by VME modules so that the data may be processed by computer. These modules are equipped with analog low-pass filters of corner frequency (-3 dB), 1.0 kHz. Also note that this filter forms a voltage divider, decreasing output signal amplitude by 1.5 %. However, this effect is easily rectified by digital signal processing.
3) **OCM Power Supply Controllers**: presently, there is a mix of VME and RS-232 based devices interfaced with the OCM power supplies. This document only considers VME-based control, as the intent is to eventually fully migrate to this technology. However, it should be noted that power supply feedback is exclusively via serial channels.
4) **Digital I/O Modules**: model VMIC-2536. These slave modules have one input and one output register, each being 32-bits in width. At present only the output registers are used, passing DAC settings to the OCM power supply controllers. The input registers were intended to be utilized in a "hand-shaking" communication protocol with the power supply controllers, but that protocol is not currently implemented [3].
5) **Digitization Modules**: model ICS-110BL sampling ADC. These modules digitize the signals produced by the Bergoz BPM modules. Each ADC features 32 input channels of 16 or 24-bit precision, sampling rates from 1 to 108 kHz, on-board storage in the form of a 128 kB FIFO, and interrupt generation when the FIFO is half-full. Unfortunately, the amount of data actually present in the FIFO may only be discovered in three special cases: FIFO empty, FIFO ½-full, and FIFO full. The impact of this constraint on software design is discussed elsewhere in this document.
6) **VME Device Access**: device driver software controlling the standard sis1100/3100 VME-PCI bridge used in several CLS facility systems. The sis3100 VME master/controller is optically connected by fiber to a sibling PCI card, the sis1100. This hardware permits some latitude in VME crate location, with respect to the PCI-side controller. More importantly, it is also possible for a single computer with multiple PCI slots to control multiple VME masters. Both are key features of the SROC architecture.
7) **EPICS Interface**: consists of custom device support which collects digitized beam-position samples, averages those samples, and then stores them in an analog-input-array record (aai). This device support also abstracts away the process of setting OCM power supply DAC values by representing each power supply channel as an analog-out (ao) record. Finally, a state-notation language (SNL) program is used to distribute the aai-record values to analog-in (ai) records representing the individual x and y coordinates produced by each BPM set.
8) **Matlab**: a Matlab application (CLSORB), originally developed at SLAC, is used to perform DC-corrections on the particle beam orbit [4]. This application utilizes Matlab Channel Access (MCA) libraries to communicate with the EPICS interface. CLSORB fetches data from the BPM process-variables, calculates the OCM field strengths required to correct the beam orbit at that location, and then writes new values to the OCM power supply process-variables.

3.0 BPM Data Processing

Currently, digital signal processing (DSP) performed on the BPM data is very basic: the data are simply averaged. While this process does not constitute a *moving average filter*, it should be noted that, in the time-domain, a moving average filter is the optimal choice for reducing white

noise while retaining the sharpest step response. The amount of noise reduction is equal to the square-root of the number of points in the average. However, in the frequency domain, such a filter is essentially a low-pass filter with slow roll-off and poor stop-band attenuation [5].

The quality of BPM positional data produced by the previous-generation of SROC system was one area where improvements could be made. This data fuels the orbit-correction algorithm, so improvements in data quality would result in more precise orbit determination and therefore, more effective control of orbit stability.

The data quality problems of the Linux-based SROC system arise directly from its data collection algorithm. This algorithm collects BPM data acquired by each VME digitization module in a sequential manner. The first of four ADC modules is *polled* until it is determined that a ½-FIFO of data has been acquired. Acquisition is then terminated for that module, its data is extracted, and these steps are repeated over the remaining three ADC modules *in sequence*. Once data has been collected from all four ADC modules, averages are calculated for each channel per ADC data-set. These averages are then passed up to the EPICS interface for distribution.

The most immediate impact of this algorithm is a detrimental effect on that system's dead time[1]. This effect may be estimated using the timeline of Figure 2, depicting one period of the SROC data collection and processing algorithm.

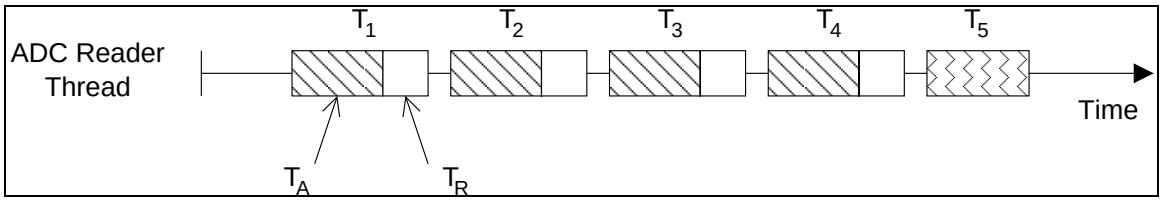

Figure 2 Timeline of SROC readout sequence.

In this figure, $T_1=T_2=T_3=T_4$, and $T_1=T_A+T_R$, where T_A is the duration spent acquiring data by a single ADC and T_R is the duration required to read that data out of the module. Using Little's Law[2] to obtain T_A and estimating T_R from data in Appendix A:

$$T_1 \approx \frac{\left(2^4 \cdot 2^{10}\left[channels\right]\right)}{\left(2^5\left[\frac{channels}{frame}\right]\right)\cdot\left(2\cdot 10^4\left[\frac{frames}{\sec}\right]\right)} + 0.003\left[\sec\right] = 0.029\left[\sec\right]$$

T_5 is the duration required to process all acquired BPM data, and is dominated by a 0.1 second pause between subsequent invocations of this entire sequence. With this information, the dead time may now be estimated as:

$$\tau_D \approx 1 - \left(\frac{T_A}{\sum_{i=1}^{5} T_i - T_A}\right) = 1 - \left(\frac{0.026}{4\cdot(0.029)+0.1-0.026}\right) = 0.87$$

[1] Dead time is the period of time during which a system is incapable of receiving new input stimuli: all input events arriving during this time are irrevocably lost. Dead time is often expressed as the fractional portion of time that a system cannot accept new input.

[2] $N = \lambda T$, where N is the system capacity, λ is the arrival rate, and T is the departure time.

Expressed another way, this SROC system collects BPM data for approximately one second out of every ten. Because the orbit correction software (CLSORB) is using data that are poorly correlated in time, the quality of its output will suffer as a result.

4.0 RTEMS-Based SROC

Despite the best efforts of developers, time-sensitive applications deployed on general-purpose operating systems (e.g. Linux) simply cannot be guaranteed to satisfy their timing constraints: kernel operations may preempt user-space programs at any time. Given the nature of the SROC system, it seems logical to implement the time-sensitive components (VME access and DSP) on a real-time OS. Deterministic software task scheduling is then available, entirely controllable by the application developer.

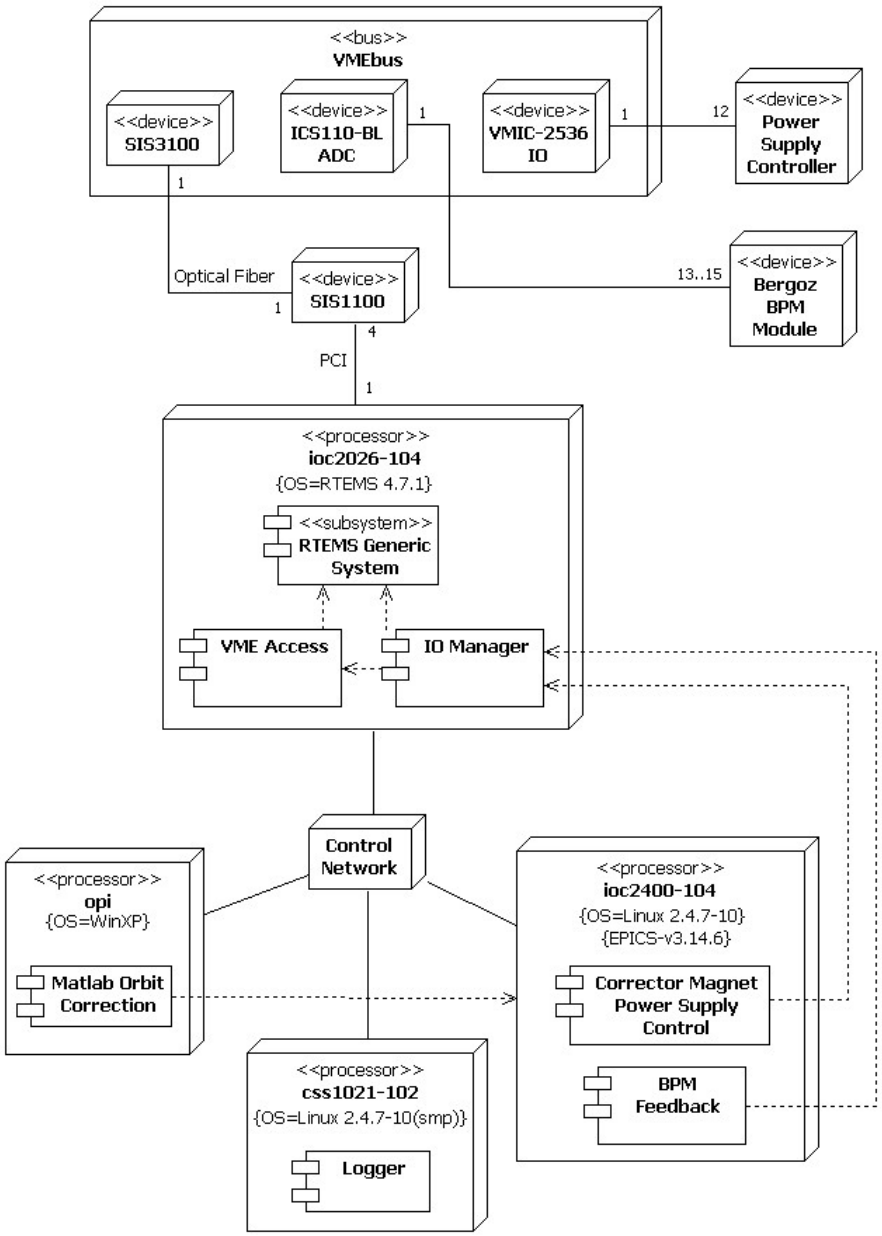

Figure 3 Component-deployment diagram of the RTEMS-based SROC system.

Utilizing RTEMS in this role, the SROC component-deployment view now appears as illustrated in Figure 3. Communication and control of VME devices are now the exclusive domain of the components deployed to the node, `ioc2026-104`, which runs the RTOS.

This architecture is able to retain much of the previous EPICS infrastructure without modification. Now, instead of requiring custom device support to communicate with the VME hardware, the RTEMS node appears as a simple "network appliance", utilizing a string-based communication protocol and requiring only standard EPICS "soft" device support. The alterations made to the EPICS interface component of the SROC system are detailed in Appendix B.

The only change made to the VME hardware configuration was a reduction of the sampling frequency of the ICS-110BL modules from 20 kHz to 10 kHz. This has the obvious effect of halving data volume requiring transport *and* processing, while still over-sampling the analog BPM data. In fact, given that the ICS-110BL modules themselves over-sample at 128-times for sampling rates below 50 kHz, the rate of the SROC system may be further reduced if needed (thereby gaining more processing time).

4.1 Implementation Details

The RTEMS processing node is responsible for direct control and communication with the VME components of the SROC system and also for averaging the acquired BPM data. This node's interaction with the CLS control system is limited to two operations:

1) transmitting the BPM data averages to an EPICS IOC for distribution, and
2) receiving OCM corrections distributed by the CLSORB application

As illustrated in Figure 3, the RTEMS node hosts three software components:

1) VME Device Access
2) Generic System
3) I/O Manager

As the VME Device Access component has been previously discussed, only items 2) and 3) are outlined in the following discussion.

4.1.1 Generic System

The RTEMS Generic System (GeSys) component is a SLAC-developed package and is responsible for initializing the RTEMS run-time environment and core services including the networking stack, TFTP and in-memory filesystems (IMFS; essentially a ramdisk), clock synchronization via NTP, and network-based logging via the *syslog* facility.

Most importantly, this component is also responsible for creating an environment conducive to loading and linking object files into an executing RTEMS system. This capability is made possible through the dependency of GeSys on another SLAC-produced package known as *cexp*.

The C-expression interpreter (cexp) is a library of utilities, offering features such as:

1) **Expression Interpreter**: also known as a *shell*, this utility permits the interpretation and invocation of arbitrary C-language expressions. While not as full-featured as the *bash* or *csh* shells, cexp may be used interactively (via console or telnet session) or autonomously, to parse and interpret scripts.
2) **Symbol-Table Management**: although an application's symbol-table must be produced and made accessible to cexp, doing so permits access to *all* symbol (variables

and functions) in the execution environment. Tools are provided by both Cexp and GeSys for creating a system symbol-table.

3) **Object-File Linking**: permits run-time loading and linking of executable code into an RTEMS system. Cexp resolves undefined object-file symbols using the system symbol-table. Newly loaded object-file symbols are added to the system symbol-table, thereby permitting Cexp to track dependencies.

Together, GeSys and cexp provide a flexible base-system from which to launch arbitrary applications. This flexibility is particularly felt during the software commissioning phase, where developmental agility is much needed.

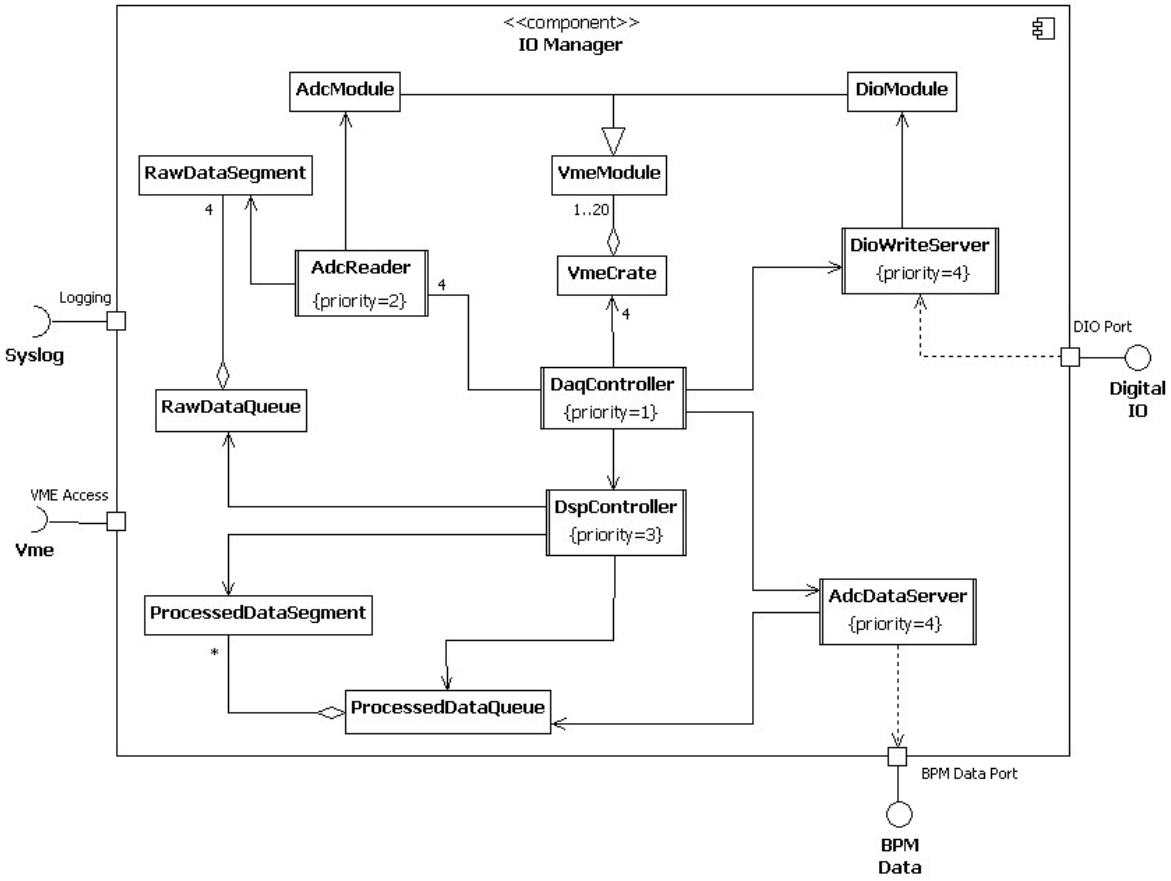

Figure 4 Concurrency-resource view of the I/O Manager. Indicated priorities are relative only.

4.1.2 I/O Manager

The structural features of the I/O Manager are illustrated by the concurrency-resource view shown in Figure 4. Within this component, it is the *DaqController* thread that is primarily responsible for orchestrating the data acquisition operations.

Unlike the polling-based design of the previous generation, this generation of SROC system is interrupt-driven, implementing the *Rendezvous* design pattern: the DaqController initiates data acquisition on all ICS-110BL modules near simultaneously, and blocks waiting on the rendezvous

condition that ½-Full FIFO interrupts have been received from *all* ADC modules. Once this condition is met, the DaqController continues execution, halts ADC data acquisition, and then directs four *AdcReader* threads to obtain the BPM data from each ADC before again becoming blocked. This time, the DaqController awaits a rendezvous on the condition that *all* AdcReader threads have completed their data transfers. Each AdcReader will place its data on a message-queue, thereby notifying the *DspController* that there is work for it, before rendezvousing with the DaqController. At this point the acquisition cycle begins again. This algorithm is graphically portrayed in Figure 5.

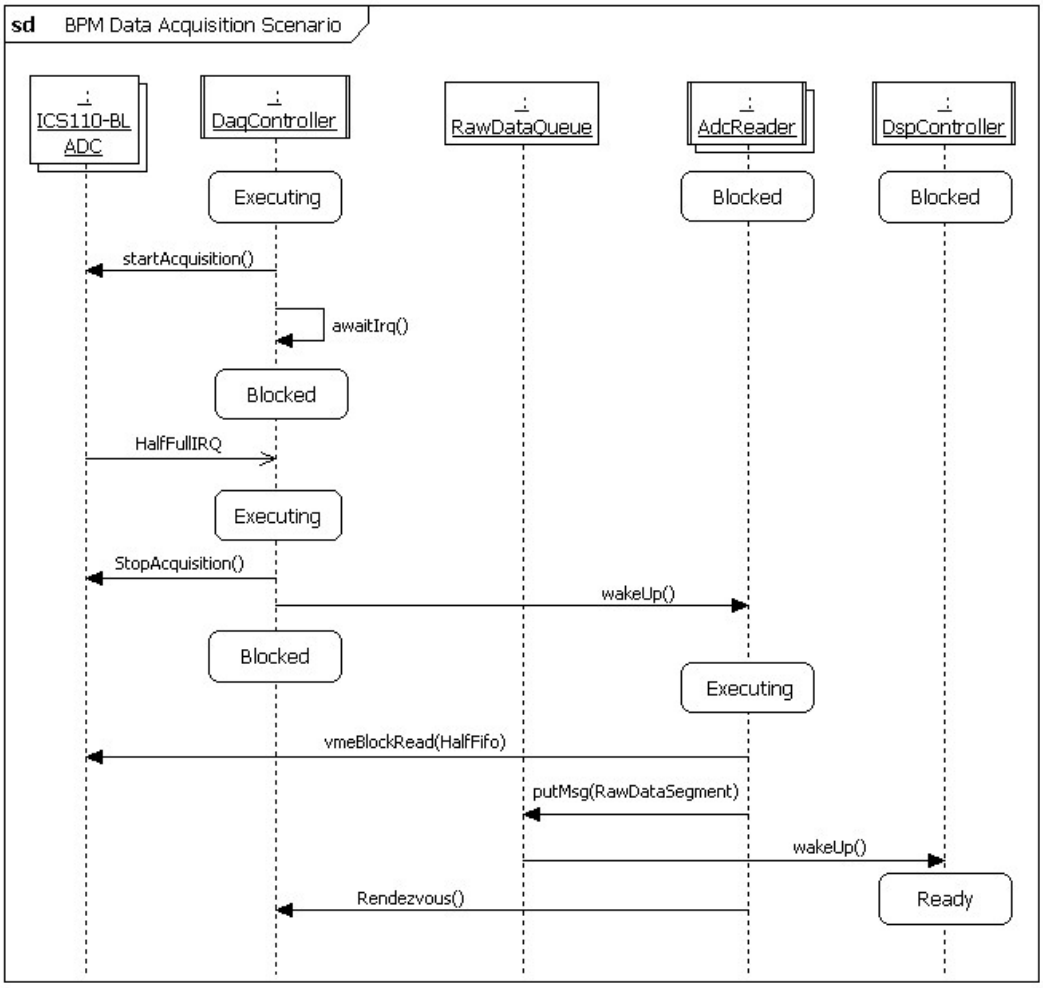

Figure 5 Sequence diagram of the I/O Manager's data collection algorithm.

Only when the DaqController and AdcReaders are blocked may the DspController thread average the acquired BPM data. This thread simply maintains a running sum of digitized BPM channels until it has amassed a compile-time specified number of samples (currently 5000 samples, equal to ½-second of data). At that time, the thread will perform averaging, scaling, and sorting operations, before placing the processed data on another message-queue where it will be retrieved by the *AdcDataServer* and transmitted to the EPICS-host, `ioc2400-104`. The operations described here are illustrated in Figure 6.

The *DioWriteServer* thread (see Figure 4) is the context which receives OCM corrections from the EPICS-host. OCM power supply corrections are defined by a 3-tuple: VME crate identity, power supply channel, and channel DAC value. This information is passed from EPICS to the

DioWriteServer as plain-text strings. The DioWriteServer simply applies the corrections in the order with which they were received. This area of the SROC system is one where improvements could easily be made (see section 5.0).

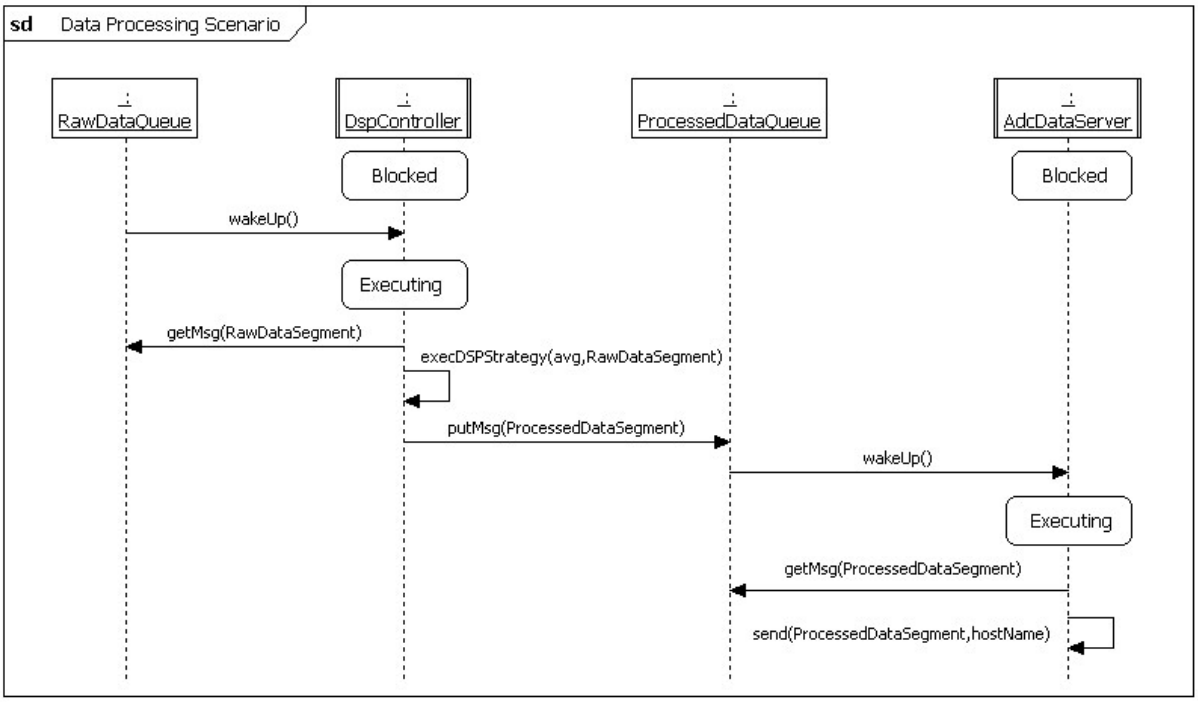

Figure 6 Sequence diagram of data processing operations.

4.2 DAQ Performance

Given that the previous generation SROC system presents new averages to the control system (EPICS) interface at the conclusion of each data acquisition cycle, that the number of samples in those averages is 254, and that the number of samples present in the new system is 5000 per average, it is expected that noise in the BPM data will be reduced by a factor of approximately:

$$\sqrt{\frac{5000}{254}} = 4.4$$

The dead time of the RTEMS-based system can also be shown to be much improved, based on the timeline of Figure 7. Using this figure, and Little's Law to first determine T_A:

$$T_1 \approx \left(\frac{2^4 \cdot 2^{10} \left[channels \right]}{\left(2^5 \left[\frac{channels}{frame} \right] \right) \cdot \left(10^4 \left[\frac{frames}{\sec} \right] \right)} \right) + 0.003 \left[\sec \right] = 0.0542 \left[\sec \right]$$

The dead time of the RTEMS-based SROC system may now be estimated as:

$$\tau_D \approx \frac{T_R}{T_1} = \frac{0.003\,[\text{sec}]}{0.0512\,[\text{sec}] + 0.003\,[\text{sec}]} = 0.05$$

This represents an improvement in DAQ dead time by a factor of approximately 16.

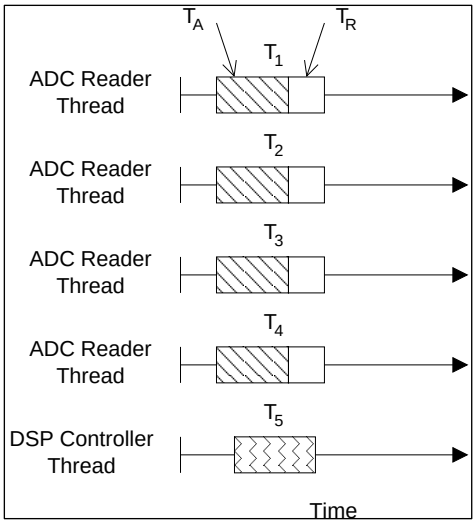

Figure 7 Timeline of RTEMS-based data acquisition sequence.

This improvement is attributable to the use of concurrency in the design. In particular, assigning independent threads to the readout of each ADC and to handle the DSP workload has proven to be effective in capitalizing on the intrinsically concurrent nature of the problem. However, at this point it is unclear what effect the improvement to the system's dead time will provide in concert with the reduction in BPM noise (is it additive ? multiplicative ?).

4.3 Implementation Issues

Unfortunately, the benefit of using RTEMS in this role comes at the expense of development and debugging ease. This is typically the case when developing cross-platform applications: i.e. using one architecture (Linux) to host the development/debugging environment of another (RTEMS). However it is possible to debug the target x86 machine via the GNU Debugger's (gdb) remote protocol, over either Ethernet or serial (RS-232) media. This capability depends on another package developed at SLAC, the RTEMS gdb-stub.

The sis1100/3100 VME-PCI interface ships with drivers supporting only the Linux and NetBSD operating systems. Despite this, an RTEMS-port of the device driver was available as a product of work previously done by this document's author. However, that RTEMS driver was produced without the capability to support multiple sis1100 cards from a single host. This capability has since been added.

Another related issue, stemming from the use of multiple sis1100 cards in single host, is that of interrupt vector sharing. The RTEMS pc386 board support package has no concept, or support, for interrupt sharing. A work-around for this problem was found, with the solution confined to the source code of the sis1100 RTEMS driver. However, this solution is subject to one caveat: devices sharing an interrupt resource must all be sis1100 devices. This is easily achieved with proper attention to the physical arrangement of PCI devices in the host.

Although RTEMS support was available for the on-board Ethernet interface of ioc2026-104, problems were experienced with it, making it unsuitable for use [6]. A PCI-based Ethernet card known to work properly under RTEMS was substituted into the system as a work-around to the problem. Incidentally, the driver for this NIC was also produced as part of the same work that produced RTEMS-port of the sis1100/3100 driver.

5.0 Remaining Issues

1) *System-level* attributes such as ADC sampling rate and average sample-size are compile-time constants. The next phase of SROC system development should present such attributes as process-variables, thereby permitting dynamic adjustment by machine operators.

2) Presently, those process-variables representing BPM x and y-coordinates do not conform to facility naming conventions. They are of the form, `1401-01:BPMx:val`, whereas the correct form would be `BPM1401-01:x`. The EPICS components resident on node ioc2400-104 and the source code of CLSORB would require updating and testing to discover any bugs arising from the PV name changes.

3) The application of more advanced digital filtering to the BPM data-stream may be an avenue worth exploring, but exactly what type of additional filtering may be useful and its performance impact on existing systems is presently unclear. This area of investigation is driven by curiosity, not necessity.

4) Currently, corrections are sequentially applied to each OCM in the order that the corrections are delivered to the I/O Manager, as demonstrated in the following pseudo-code:

```
foreach correction
      setPowerSupply(vmeCrate,channel)
      setPowerSupply(vmeCrate,data)
      setPowerSupply(vmeCrate,LATCH)
      setPowerSupply(vmeCrate,UPDATE)
```

The algorithm updating the OCM power supplies would benefit from parallelization, such that the channel and DAC data are set and latched for each correction *and then* update all the new settings:

```
getPowerSupplyCorrections()

foreach vmeCrate
      foreach channel
            setPowerSupply(channel)
            setPowerSupply(data)
            setPowerSupply(LATCH)
foreach vmeCrate
            setPowerSupply(UPDATE)
```

This approach carries the benefit that power supply set-points for all orbit correction magnets take effect near simultaneously. Note, implicit in this algorithm is the requirement that OCM power supply corrections be delivered to the I/O Manager in a batch format: *all* corrections must be delivered before the algorithm may commence.

5) The method described above for affecting near-simultaneous OCM adjustment becomes a practical necessity in any future SROC design incorporating *dynamic* orbit control. Also known as *fast*, or AC orbit correction, this type of design would implement a correction bandwidth on the order of 100 Hz: BPM data are collected and OCM set-points are calculated and distributed with a periodicity on the order of 10 ms.

Development of an AC orbit control system dictates that the DaqController's impetus be revised from interrupt-driven to *period-driven*. This paradigm shift is mandated primarily by the constraints associated with ICS-110BL modules. Namely, their inability to provide detailed information on the volume of data in their FIFO buffers. A period-driven design carries with it much concern for the time-stability requirements imposed on it. Also, there is a high degree of uncertainty associated with the use of EPICS and Matlab in a system with such stringent timing requirements.

6) The role of EPICS in the evolution of the SROC system deserves much attention. It is certainly a requirement at this point, due to the CLSORB-EPICS dependency. One area worth exploring would be to examine the impact of relocating the services of the EPICS node (`ioc2400-104`) to the data acquisition and OCM-control node (`ioc2026-104`). While this idea somewhat simplifies the design of the SROC system, the performance impact and developmental complexities remain significant uncertainties.

6.0 References

[1] Dallin, L. 2000. *SR1 Dynamic Orbit Correction*. CLS Document 5.2.69.4 Rev. 0

[2] Berg, R. 2004. *Orbit Control for the Canadian Light Source*. EPAC Proceedings

[3] Berg, R. 2003. *Orbit Corrector Power Supply Communications Protocol for SR1*. CLS Document 7.2.39.7 Rev. 0

[4] Zhang, H. and Berg, R. *CLSORB: Slow SR Orbit Control in Matlab*. CLS Document 7.9.X.X Rev. 0

[5] Smith, S.W. *The Scientist and Engineer's Guide to Digital Signal Processing*. Available online: http://www.dspguide.com

[6] Harrison, R. *SR Orbit Correction Design*. CLS Document (not issued for review).

Appendix A: ICS-110BL Block Transfer Rate Measurements

Early on in the test-driven development of the new SROC system, benchmarking was performed to determine the duration of VME-to-PCI block transfers (BLT) as a function of payload size for the ICS-110BL modules. It must be noted that the data obtained here pertain only to the ICS-110BL/sis1100/RTEMS combination: substitution of any of those components would yield different results.

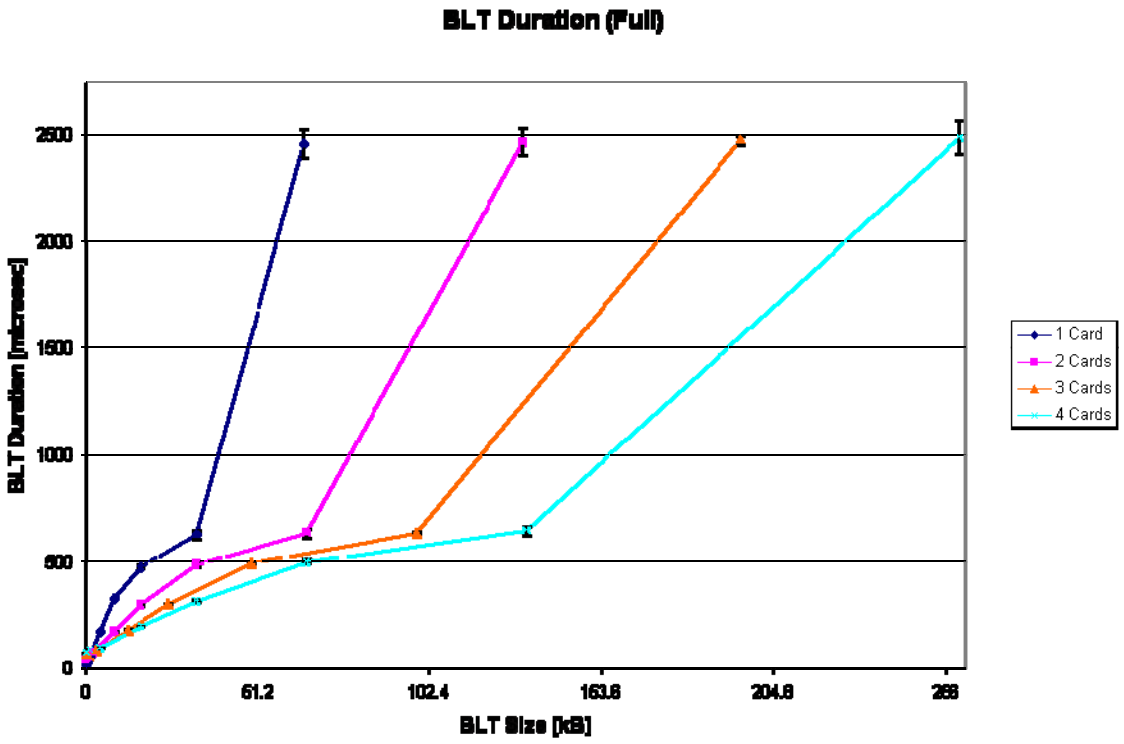

Figure 8 Block transfer duration as a function of payload size.

Timing statistics for these operations were obtained by first calibrating the time-stamp counter (TSC) of IOC2026-104 against its i8254 programmable interrupt timer (PIT). This procedure established the number of TSC counts per second. With this information, it is possible to unobtrusively determine software method durations by obtaining time-stamps at the onset and conclusion of those methods. The difference between those time-stamps is directly proportional to the duration of the operation being measured. Finally, scaling the differences by the calibration factor yields the properly dimensioned duration.

The reason behind the non-linearity of Figure 8 is presently unclear. This feature seems unlikely to arise due inter-thread synchronization as the non-linearity is present for both single and multi-card transfers. This fact alone would seem to imply a hardware-related cause.

Figure 9 is a truncated plot of the above data-set, focusing on the small payload, linear region. The solid lines in this plot reflect the least-squares fitting applied to each data set. The slopes of these equations have dimensions of microseconds per byte, while the y-intercepts may be interpreted as the duration (in microseconds) required to perform a block transfer with a null-sized payload. In effect, this is the minimum cost of invoking a BLT for the specified number of ADC cards. The transfer rate and minimum-cost data are summarized Table 1-1.

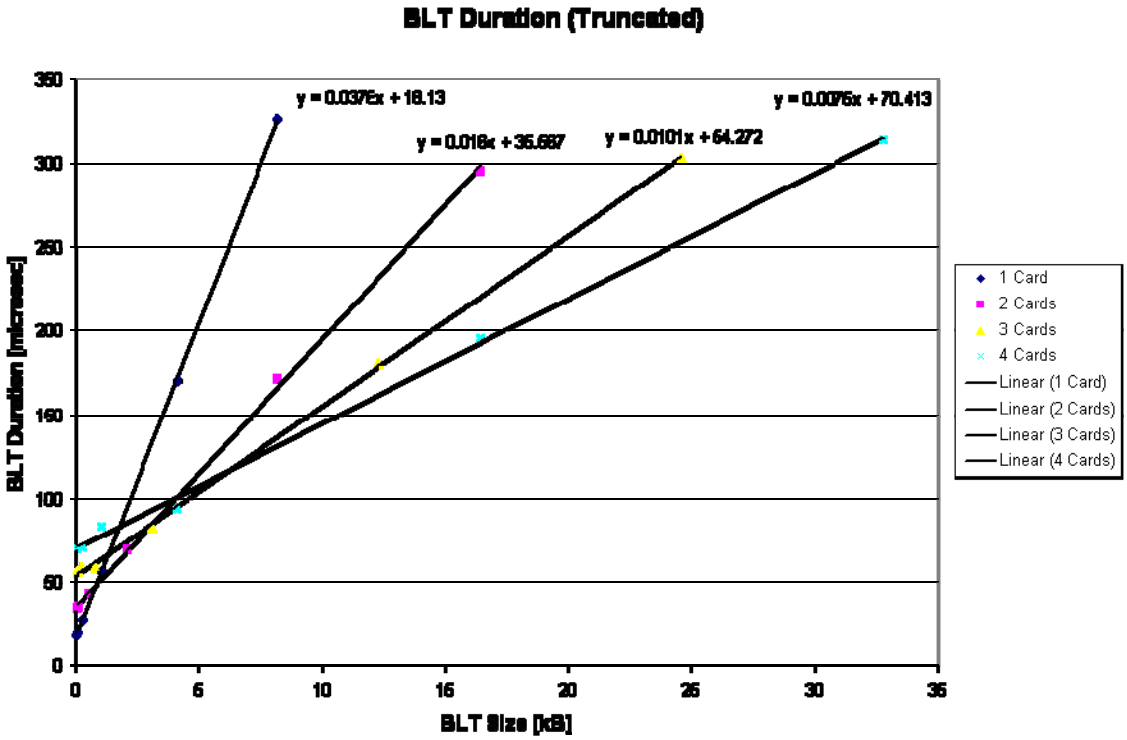

Figure 9 Truncated plot of BLT duration data set.

Number of ADC cards	BLT Rate $\left[\frac{MB}{s}\right]$	BLT Minimum Cost $[\mu s]$
1	26.6	18.1
2	62.5	35.6
3	99.0	54.3
4	132.0	70.4

Table 1-1 Summary of BLT rate benchmarking results.

To quantify the benefits of concurrent versus sequential ADC-readout techniques, a parallelization, or *speedup*-factor may be calculated. This factor is expressed as:

$$speedup = \frac{sequential\ duration}{parallel\ duration}$$

Application of this equation to the BLT duration data-set is plotted in Figure 10.

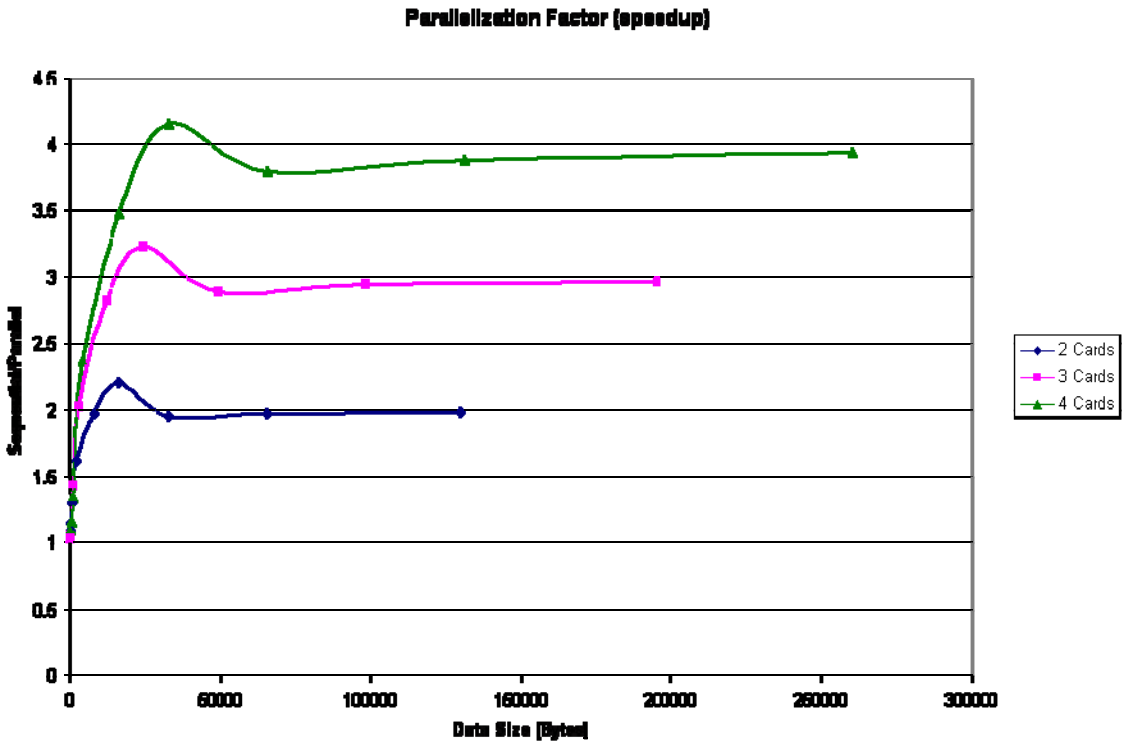

Figure 10 **Speedup factor resulting from parallelization of ADC readout operation.**

Appendix B: Modifications to the EPICS SROC components

Given the tight coupling between the CLSORB program and the control system interface it utilizes, care was exercised to ensure that interface would not require modification to operate with the RTEMS-based system changes. However, the implementation beneath that control system interface is substantially different than that utilized with the Linux-based system.

Previously, the implementation relied on custom device support to abstract and affect VME device access as EPICS process-variable operations: an analog-array was updated by the device support, reflecting the averaged BPM data values, while individual OCM DAC channels were represented by a custom record-type. Associated with this software machinery is a state-notation language (SNL) program, which monitored the analog-array for updates and distributed each field of the array into individual records representing either an x or y BPM coordinate average.

With the RTEMS-based system, communication between the EPICS I/O controller and the RTEMS node is carried out over two TCP/IP sockets (see Figure 4): one to pass BPM data back to the EPICS IOC, and one for the IOC to send OCM power supply settings to the RTEMS node.

The OCM-EPICS components rely on ASYN-record types to maintain the appearance of the RTEMS node as string-driven, network appliance. This record-type permits the network and byte-oriented data transfers required by the system.

An SNL component is also an integral part of the RTEMS-based system. This component receives the averaged BPM data over a socket and distributes that data to the individual records representing BPM coordinates. The behaviour of this SNL artifact, ocFSM.stt, is depicted in Figure 11.

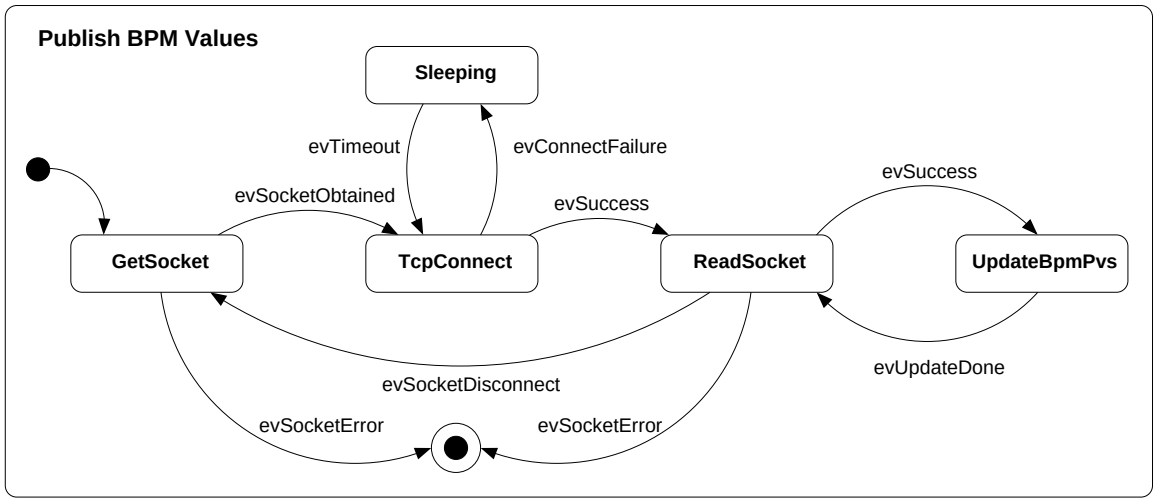

Figure 11 State diagram of the SNL program, ocFSM.stt