EDOC060

EUROGAM PROJECT

NSF DATA ACQUISITION SYSTEM

Eurogam Spectrum Servers: Online Histogrammer

Edition 0.9

December 1990

NSF Software Systems Group

UK Science and Engineering Research Council

Daresbury Laboratory

Eurogam Spectrum Servers
Online Histogrammer

David Brightly

Edition 0.9
December 1990

[As usual this document is not yet complete but is published in its present form for comments. More work needs to be done in defining functions for returning some form of compressed histogram data and for obtaining cuts and slices from histograms of dimension > 1. XDR bindings for all the procedures also need to be worked out. Author's remarks and asides are in square brackets. D.B. Monday 10 December 1990]

1 Introduction

This document is in two parts: the first part specifies the properties of spectra in general and defines generic operations on them; the second part describes the EuroGam online histogrammer hardware and shows how the operations defined in the first part can be mapped onto it. Further documents in this series will describe similar specialisations for online sorter spectra and saved disc spectra.

2 Generic spectra

2.1 Shapes, tuples, and domains

This section will describe the foundations which we will need to talk about histograms and spectra.

$$\begin{aligned}
&SHAPE == \mathrm{seq}_1(\mathbb{Z} \times \mathbb{Z})\\
&TUPLE == \mathrm{seq}_1\,\mathbb{Z}\\
&DOMAIN == \mathbb{P}\,TUPLE
\end{aligned}$$

Histograms are essentially arrays of a certain *shape*. For example, a simple one dimensional histogram might have a shape given by $\langle(0, 1024)\rangle$. This means it starts with channel 0 and has 1024 channels. A two dimensional histogram with 64 channels in the first dimension and 4096 in the second could have the shape $\langle(1, 64), (0, 4096)\rangle$. In this example the channels in the first dimension are numbered 1 to 64. A *tuple* is a sequence of numbers labelling a channel in a histogram. The tuple $\langle 23\rangle$ labels channel 23 of our first histogram and the tuple $\langle 44, 2753\rangle$ labels a channel in our two dimensional example. A *domain* is just a set of tuples. We will use domains to talk about subsets of all the channels in a histogram. Both tuples and shapes have a *dimension* defined in the obvious way:

$$\begin{array}{|l}
dimension : TUPLE \rightarrow \mathbb{N}_1 \\
shapedimension : SHAPE \rightarrow \mathbb{N}_1 \\
\hline
\forall\, t : TUPLE \bullet dimension(t) = \# t \\
\forall\, s : SHAPE \bullet shapedimension(s) = \# s
\end{array}$$

We can count the total number of channels in a histogram of a certain shape. Our first example has 1024 channels and the second has $64 * 4096$ channels:

$$\begin{array}{|l}
size : SHAPE \rightarrow \mathbb{Z} \\
\hline
\forall\, a, b : \mathbb{Z};\ s : SHAPE \bullet \\
\qquad size\ \langle (a, b) \rangle = b \land \\
\qquad size(s) = size(\langle head\ \ s \rangle) * size(tail\ \ s)
\end{array}$$

When reading or writing counts in channels we need to present them in some fixed order. In this ordering, the rightmost channel coordinate varies fastest, so we have $\langle 100 \rangle$ precedes $\langle 200 \rangle$ and $\langle 23, 3192 \rangle$ precedes $\langle 44, 1024 \rangle$.

$$\begin{array}{|l}
_ \text{ precedes } _ : TUPLE \leftrightarrow TUPLE \\
\hline
\forall\, a, b : \mathbb{Z};\ x, y : TUPLE \mid \# x = \# y \bullet \\
\qquad (\langle a \rangle \text{ precedes } \langle b \rangle \Leftrightarrow a < b) \land \\
\qquad (x \text{ precedes } y \Leftrightarrow (head\ \ x < head\ \ y \lor (head\ \ x = head\ \ y \land tail\ \ x \text{ precedes } tail\ \ y)))
\end{array}$$

We can say whether a given sequence of tuples is ordered or not:

$$\begin{array}{|l}
\text{ordered}_ : \mathbb{P}(\text{seq}\, TUPLE) \\
\hline
\forall\, s : \text{seq}\, TUPLE \bullet \\
\qquad \text{ordered } s \Leftrightarrow (\forall\, i, j : \text{dom}\, s \bullet (i < j) \Leftrightarrow s\ \ i \text{ precedes } s\ \ j)
\end{array}$$

and given a set of tuples we can order them:

$$\begin{array}{|l}
order : DOMAIN \rightarrow \text{seq}\, TUPLE \\
\hline
\forall\, d : DOMAIN;\ s : \text{seq}\, TUPLE \bullet \\
\qquad s = order(d) \Leftrightarrow \text{ran}\, s = d \land \text{ordered } s
\end{array}$$

We can also say whether a given tuple lies within a given shape. For example, we have $\langle 11, 999 \rangle$ within $\langle (1, 64), (0, 4096) \rangle$ but not $\langle 2048 \rangle$ within $\langle (0, 1024) \rangle$.

$$\begin{array}{|l}
_ \text{ within } _ : TUPLE \leftrightarrow SHAPE \\
\hline
\forall\, x, a, b : \mathbb{Z};\ t : TUPLE;\ s : SHAPE \bullet \\
\qquad (\langle x \rangle \text{ within } \langle (a, b) \rangle \Leftrightarrow 0 \leq x - a < b) \land \\
\qquad (t \text{ within } s \Leftrightarrow \langle head\ \ t \rangle \text{ within } \langle head\ \ s \rangle \land tail\ \ t \text{ within } tail\ \ s)
\end{array}$$

Given a shape we can now return the domain consisting of all the channels that lie within that shape:

$$\begin{array}{|l}
rectdomain : SHAPE \rightarrow DOMAIN \\
\hline
\forall\, s : SHAPE \bullet rectdomain(s) = \{ d : TUPLE \mid d \text{ within } s \}
\end{array}$$

Finally, if all the channels within a shape are ordered in this way we can calculate the offset of a given channel from the beginning of the sequence.

$$\begin{array}{|l}
\mathit{offset} : \mathit{TUPLE} \times \mathit{SHAPE} \to \mathbb{Z} \\
\hline
\forall\, s : \mathit{SHAPE};\ x, a, b : \mathbb{Z};\ t : \mathit{TUPLE} \bullet \\
\qquad \mathit{offset}(\langle x \rangle, \langle (a, b) \rangle) = x - a \land \\
\qquad \mathit{offset}(t, s) = \mathit{offset}(\langle \mathit{head}\ t \rangle, \langle \mathit{head}\ s \rangle) * \mathit{size}(\mathit{tail}\ s) + \mathit{offset}(\mathit{tail}\ t, \mathit{tail}\ s)
\end{array}$$

For example, we have

$$\mathit{offset}(\langle 23 \rangle, \langle (0, 1024) \rangle) = 23$$

and

$$\mathit{offset}(\langle 11, 999 \rangle, \langle (1, 64), (0, 4096) \rangle) = 10 * 4096 + 999.$$

The following relationships can be proved (by induction on the dimension of the shape s):

$$\begin{array}{l}
\forall\, x, y : \mathit{TUPLE};\ s : \mathit{SHAPE} \mid x \text{ within } s \land y \text{ within } s \bullet \\
\qquad 0 \leq \mathit{offset}(x, s) < \mathit{size}\ s \land \\
\qquad x \text{ precedes } y \Leftrightarrow \mathit{offset}(x, s) < \mathit{offset}(y, s) \land \\
\qquad \mathit{order}(\mathit{rectdomain}\ s)(\mathit{offset}(x, s) + 1) = x
\end{array}$$

The last relation just says that the *offset* function assigns to a tuple x the ordinal number of its position in the ordered domain based on the shape s (less one because sequences in Z start at one not zero, unfortunately).

2.2 Histograms and Spectra

We can now introduce histograms and spectra as data types. A histogram consists of a function from some set of tuples, i.e., channels, into the integers. This function is defined on all the channels within some shape. It is also useful to hold with each histogram some indication of its least and greatest count. For example, an interactive spectrum display program can determine how to scale the counts for display without first having to read them all. However, we don't insist that the values always tell the truth—this would be difficult to achieve within a hardware histogramming system, for instance.

$$\begin{array}{|l|}
\hline
\mathit{Histogram} \\
\hline
\mathit{shape} : \mathit{SHAPE} \\
\mathit{counts} : \mathit{TUPLE} \nrightarrow \mathbb{Z} \\
\mathit{mincount} : \mathbb{Z} \\
\mathit{maxcount} : \mathbb{Z} \\
\hline
\mathrm{dom}\ \mathit{counts} = \mathit{rectdomain}\ \mathit{shape} \\
\hline
\end{array}$$

A spectrum consists not only of counts but also of *calibrations* and *annotations*. We don't need to specify at this stage what they look like—in fact they will be opaque strings of characters associated with spectra—so we will treat them for the time being as a given data type. But we will need a reserved value from this type to denote the empty string, i.e., an undefined calibration or annotation. We will also need to distinguish between histograms of different *precision*, i.e., the number of bits required to represent the counts. *[need to think out the semantics of precision further]*

$$[STRING]$$
$$PRECISION ::= bit16 \mid bit32$$

$$
\begin{array}{|l}
maxannotations : \mathbb{N} \\
maxhistograms : \mathbb{N} \\
maxcalibrations : \mathbb{N} \\
NULL : STRING \\
\hline
maxannotations = 16 \\
maxcalibrations = 16 \\
maxhistograms = 16
\end{array}
$$

A spectrum is specified as follows:

$$
\begin{array}{|l|}
\hline
\textit{Spectrum} \\
precision : PRECISION \\
sshape : SHAPE \\
arrays : 1 \mathrel{..} maxhistograms \pfun Histogram \\
calibrations : 1 \mathrel{..} maxcalibrations \pfun STRING \\
annotations : 1 \mathrel{..} maxannotations \pfun STRING \\
\hline
\forall i : \operatorname{dom} arrays \bullet (arrays\ i).shape = sshape \\
\operatorname{dom} calibrations \subseteq 1 \mathrel{..} shapedimension\ sshape \\
\hline
\end{array}
$$

A spectrum thus consists of zero or more histograms, calibrations, annotations, all identified by small integers. These will all have some conventional use—for example, the first histogram would contain the principal counts of the spectrum and the second (if it existed) might contain an "error" histogram if the principal counts had been obtained (say) by some background subtraction operation. Similarly the first annotation might by convention be a descriptive title for the spectrum included by application programs in formatted output, and so on. All we require at the moment is to establish an upper limit on the numbers of such objects making up a spectrum, as this will guide implementation decisions. Note, however, that the histograms must all have a common shape, and each calibration (if it exists) is associated with a specific dimension.

We will need a data type for returning information about a spectrum:

$$
\begin{array}{|l|}
\hline
\textit{Attributes} \\
s : SHAPE \\
p : PRECISION \\
\hline
\end{array}
$$

2.3 Spectrum store

Users will want to reference spectra using meaningful names. We can envisage at least three sources of spectra: online histogramming, online EbyE sorting, and saved spectra on disc. Each spectrum server manages a "spectrum store", i.e., a directory structure which maps names to spectra. In the case of online histogramming and sorting the store need offer no more than a single level naming scheme, but in the case of saved spectra on disc the store can be identified with (part of) the host system filestore and will typically offer a multi-level directory structure. Both online and saved spectrum directories can be integrated by adopting a naming convention which reserves specific directory names for histogrammers and sorters and which enables client programs to identify which server supports a spectrum of any given name:

$$
\begin{array}{lll}
\textit{name} & ::= & /\texttt{histogrammer}n/\textit{simplename} \\
 & | & /\texttt{sorter}n/\textit{simplename} \\
 & | & /\textit{simplename}/\ldots/\textit{simplename} \\
\textit{simplename} & ::= & \text{any printable name without / or blank} \\
n & ::= & \textit{integer}
\end{array}
$$

[in a multiple front end system should there be a leading `/eg1`*, say, to identify the frontend, or should this be information in the client context?]*

However, for our current purposes we need not be too specific about directories and stores and will just assume the following types:

$$
\begin{array}{l}
[NAME, SIMPLENAME] \\
DIRECTORY == SIMPLENAME \nrightarrow Spectrum \\
STORE == NAME \nrightarrow Spectrum
\end{array}
$$

Each spectrum server will operate a single store of spectra and in the case of online spectrum servers this will consist of a single directory.

SpectrumStore

$$
store : STORE
$$

Given a directory within the store we must be able to obtain a list of the spectra within it (analogous to the unix `ls` command):

ReadDirectory

$$
\begin{array}{l}
dir? : DIRECTORY \\
nn! : \operatorname{iseq} SIMPLENAME \\
\hline
\operatorname{ran} nn! = \operatorname{dom} dir?
\end{array}
$$

and also a list of spectra together with their attributes (analogous to `ls -l`):

ReadDirectoryWithAttributes

$$
\begin{array}{l}
dir? : DIRECTORY \\
nana! : \operatorname{iseq}(SIMPLENAME \times Attributes) \\
\hline
\forall\, n : SIMPLENAME;\ s : Spectrum;\ a : Attributes \bullet \\
(n, a) \in \operatorname{ran} nana! \Leftrightarrow s = dir?(n) \wedge a.s = s.sshape \wedge a.p = s.precision
\end{array}
$$

2.4 Generic operations on spectra

This section will outline the operations that all spectrum servers have to support. Later on we will specialise these operations for the EuroGam histogrammer.

$$
\begin{array}{|l}
\hline
\textit{CreateSpectrum} \\
\Delta \textit{SpectrumStore} \\
spectrum : Spectrum \\
name? : NAME \\
s? : SHAPE \\
p? : PRECISION \\
h : Histogram \\
\hline
name? \notin \operatorname{dom} store \\
spectrum.precision = p? \\
spectrum.sshape = s? \\
spectrum.calibrations = \varnothing \\
spectrum.annotations = \varnothing \\
spectrum.arrays = \{1 \mapsto h\} \\
h.shape = s? \\
\operatorname{ran} h.counts = \{0\} \\
h.mincount = h.maxcount = 0 \\
store' = store \oplus \{name? \mapsto spectrum\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{DeleteSpectrum} \\
\Delta \textit{SpectrumStore} \\
name? : NAME \\
\hline
name? \in \operatorname{dom} store \\
store' = \{name?\} \ntriangleleft store \\
\hline
\end{array}
$$

Note that a brand new spectrum comes with an already zeroed first histogram. This is convenient for client programs of the online spectrum servers, but in the case of large disc spectra it is probably desirable to optimise out the writing of many zero channels—this happens automatically with the unix file system, for example.

The following two schemas provide convenient abbreviations to be used in specifying operations on spectra: the first is used in "read-only" operations and the second in those that alter the contents of the spectrum store.

$$
\begin{array}{|l}
\hline
\textit{AccessSpectrum} \\
\Xi \textit{SpectrumStore} \\
n? : NAME \\
s : Spectrum \\
\hline
n? \in \operatorname{dom} store \\
s = store\ n? \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{ModifySpectrum} \quad}} \\
\Delta \mathit{SpectrumStore} \\
n? : \mathit{NAME} \\
s, s' : \mathit{Spectrum} \\
\hline
n? \in \operatorname{dom} \mathit{store} \\
s = \mathit{store}\ n? \\
s'.\mathit{sshape} = s.\mathit{sshape} \\
\mathit{store}' = \mathit{store} \oplus \{n? \mapsto s'\} \\
\hline
\end{array}
$$

The following operation returns the attributes of a spectrum—its shape and precision:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{InquireAttributes} \quad}} \\
\mathit{AccessSpectrum} \\
n? : \mathit{NAME} \\
a! : \mathit{Attributes} \\
\hline
a!.s = s.\mathit{sshape} \land a!.p = s.\mathit{precision} \\
\hline
\end{array}
$$

We can now give schemas describing the operations of reading and writing the component parts of a spectrum.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{ReadHistogram} \quad}} \\
\Xi \mathit{SpectrumStore} \\
\mathit{AccessSpectrum} \\
i? : \mathbb{N} \\
s? : \mathit{SHAPE} \\
h : \mathit{Histogram} \\
z! : \operatorname{seq} \mathbb{Z} \\
\hline
i? \in \operatorname{dom} s.\mathit{arrays} \\
h = s.\mathit{arrays}\ i? \\
\mathit{rectdomain}\ s? \subseteq \mathit{rectdomain}\ h.\mathit{shape} \\
z! = h.\mathit{counts} \circ (\mathit{order}(\mathit{rectdomain}\ s?)) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{WriteHistogram} \\
\hline
\Delta SpectrumStore \\
ModifySpectrum \\
i? : \mathbb{N} \\
s? : SHAPE \\
z? : \operatorname{seq} \mathbb{Z} \\
h, h' : Histogram \\
\hline
i? \in \operatorname{dom} s.arrays \\
h = s.arrays\ i? \\
rectdomain\ s? \subseteq rectdomain\ h.shape \\
h'.counts = h.counts \oplus \{ j : \operatorname{dom} z? \bullet order(rectdomain\ s?) j \mapsto z?\ j \} \\
h'.shape = h.shape \\
s'.arrays = s.arrays \oplus \{ i? \mapsto h' \} \\
s'.annotations = s.annotations \\
s'.calibrations = s.calibrations
\end{array}
$$

$$
\begin{array}{l}
\textbf{ZeroHistogram} \\
\hline
\Delta SpectrumStore \\
ModifySpectrum \\
i? : \mathbb{N} \\
s? : SHAPE \\
h, h' : Histogram \\
\hline
i? \in \operatorname{dom} s.arrays \\
rectdomain\ s? \subseteq rectdomain\ h.shape \\
h'.counts = h.counts \oplus \{ t : rectdomain\ s? \bullet t \mapsto 0 \} \\
h'.shape = h.shape \\
h'.mincount = h'.maxcount = 0 \\
s'.arrays = s.arrays \oplus \{ i? \mapsto h' \} \\
s'.annotations = s.annotations \\
s'.calibrations = s.calibrations
\end{array}
$$

$$
\begin{array}{l}
\textbf{ReadExtrema} \\
\hline
\Xi SpectrumStore \\
AccessSpectrum \\
i? : \mathbb{N} \\
h : Histogram \\
min!, max! : \mathbb{Z} \\
\hline
i? \in \operatorname{dom} s.arrays \\
h = s.arrays\ i? \\
min! = h.mincount \\
max! = h.maxcount
\end{array}
$$

$$
\begin{array}{l}
\textbf{WriteExtrema} \\
\hline
\Delta SpectrumStore \\
ModifySpectrum \\
i? : \mathbb{N} \\
min?, max? : \mathbb{Z} \\
h, h' : Histogram \\
\hline
i? \in \operatorname{dom} s.arrays \\
h'.shape = h.shape \\
h'.counts = h.counts \\
h'.mincount = min? \\
h'.maxcount = max? \\
s'.arrays = s.arrays \oplus \{i? \mapsto h'\} \\
s'.annotations = s.annotations \\
s'.calibrations = s.calibrations \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{ReadCalibration} \\
\hline
\Xi SpectrumStore \\
AccessSpectrum \\
c! : STRING \\
i? : \mathbb{N} \\
\hline
i? \in \operatorname{dom} s.calibrations \\
c! = s.calibrations\ i? \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{WriteCalibration} \\
\hline
\Delta SpectrumStore \\
ModifySpectrum \\
c? : STRING \\
i? : \mathbb{N} \\
\hline
i? \in \operatorname{dom} s.calibrations \\
s'.sshape = s.sshape \\
s'.arrays = s.arrays \\
s'.annotations = s.annotations \\
s'.calibrations = s.calibrations \oplus \{i? \mapsto c?\} \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{ReadAnnotation} \\
\hline
\Xi SpectrumStore \\
AccessSpectrum \\
a! : STRING \\
i? : \mathbb{N} \\
\hline
i? \in 1 \mathrel{..} maxannotations \\
a! = s.annotations\ i? \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{WriteAnnotation} \\
\hline
\Delta SpectrumStore \\
ModifySpectrum \\
a? : STRING \\
i? : \mathbb{N} \\
\hline
i? \in 1 \mathbin{..} maxannotations \\
s'.sshape = s.sshape \\
s'.arrays = s.arrays \\
s'.annotations = s.annotations \oplus \{i? \mapsto a?\} \\
s'.calibrations = s.calibrations \\
\hline
\end{array}
$$

3 EuroGam online histogrammer

3.1 Histogrammer memory

The previous sections have described spectra in general. We now come to specific properties of the EuroGam online histogrammer. Let us assume that the histogramming memory occupies a contiguous block of 32 bit words in the VME address space between word address *RMbase* and $RMbase + RMrange - 1$ inclusive. It turns out to be easier to model the memory as a sequence of words rather than bytes, so all our "addresses" have to be multiplied by four to get actual VME addresses.

$$
\begin{array}{l}
VMElimit : \mathbb{N} \\
RMbase : \mathbb{N} \\
RMrange : \mathbb{N} \\
\hline
VMElimit = 1024 * 1024 * 1024 \\
0 \leq RMbase \land RMbase + RMrange \leq VMElimit
\end{array}
$$

We can model the actual histogramming memory with a function which returns the contents of a memory word, given its address.

$$
\begin{array}{l}
\textbf{RealMemory} \\
\hline
memory : \mathbb{N} \rightarrow \mathbb{Z} \\
\hline
\operatorname{dom} memory = RMbase \mathbin{..} RMbase + RMrange - 1 \\
\hline
\end{array}
$$

All our histograms will occupy real memory so we need a way of managing the space. We will do this with a pair of functions which record the base address (relative to *RMbase*) and number of words occupied of each histogram.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{HistogramMemory} \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
\mathit{membase} : \mathit{Histogram} \nrightarrow \mathbb{Z} \\
\mathit{memrange} : \mathit{Histogram} \nrightarrow \mathbb{Z} \\
\mathit{memused} : \mathit{Histogram} \nrightarrow \mathbb{P}\,\mathbb{Z} \\
\mathrm{memfree}\_ : \mathbb{P}(\mathbb{Z} \times \mathbb{Z}) \\
\hline
\mathrm{dom}\,\mathit{membase} = \mathrm{dom}\,\mathit{memrange} \\
\\
\forall\, h : \mathit{Histogram} \bullet \\
\quad h \notin \mathrm{dom}\,\mathit{membase} \Rightarrow \mathit{memused}(h) = \varnothing \wedge \\
\quad h \in \mathrm{dom}\,\mathit{membase} \Rightarrow \mathit{memused}(h) = \mathit{membase}(h) \mathinner{..} \mathit{membase}(h) + \mathit{memrange}(h) - 1 \\
\\
\forall\, m, r : \mathbb{Z} \bullet \\
\quad \mathrm{memfree}(m, r) \Leftrightarrow (\forall\, h : \mathrm{dom}\,\mathit{membase} \bullet \\
\quad\quad (m \mathinner{..} m + r - 1) \cap \mathit{memused}\ h = \varnothing) \wedge 0 \leq m < m + r \leq \mathit{RMrange} \\
\hline
\end{array}
$$

To simplify discussion of memory management *memused* returns the set of memory addresses occupied by a histogram, and $\mathrm{memfree}(m, r)$ tells us whether there is a block of r unused words starting at address m.

3.2 Histogrammer card

In this section we describe the EuroGam histogrammer card as outlined in [1]. We need to talk about enabling and disabling aspects of the card's function:

$$\mathit{STATE} ::= \mathit{enabled} \mid \mathit{disabled}$$

We need to refer to the data acquisition channel "addresses" which label conversions received by the histogrammer. We will call these *tags* in order to distinguish them from histogramming memory addresses.

$$[\mathit{TAG}]$$

We can now describe the histogrammer card itself:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{HistogrammerCard} \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
\mathit{state} : \mathit{STATE} \\
\mathit{tagstate} : \mathit{TAG} \rightarrow \mathit{STATE} \\
\mathit{tagbase} : \mathit{TAG} \nrightarrow \mathbb{Z} \\
\mathit{baseaddress} : \mathbb{N} \\
\hline
\end{array}
$$

We have an overall *state* which determines whether words are histogrammed or not and a function *tagstate* which tells us whether a given tag is to be histogrammed or not. Note that *tagstate* is a total function—it must be defined for all possible tags. The function *tagbase* and the constant *baseaddress* determine (indirectly) the base address in memory of the histogram generated by each tag, These objects directly model the hardware features described in [1]. In particular, *tagstate* and *tagbase* correspond to the histogrammer's "look-up table".

The histogrammer card generates addresses to be incremented by shifting and "or"-ing. If we use the infix function $\_\mathrm{bitor}\_$ to denote bitwise "or" on two's complement integers we can specify the operation of the histogrammer card when presented with a tag $t?$ and a data word $d?$ [1, section 3.3].

$$
\begin{array}{|l}
\hline
\textit{Increment} \\
\Xi \textit{HistogrammerCard} \\
\Delta \textit{RealMemory} \\
t? : \textit{TAG} \\
d? : \mathbb{N} \\
\hline
\textit{state} = \textit{enabled} \land \textit{tagstate}\ t? = \textit{enabled} \Leftrightarrow \\
\quad (\exists\, a : \mathbb{N} \bullet \\
\quad\quad a = \textit{baseaddress} \operatorname{bitor} (\textit{tagbase}\ t? * 1024) \operatorname{bitor} d? \land \\
\quad\quad \textit{memory}' = \textit{memory} \oplus \{a \mapsto \textit{memory}(a) + 1\}) \\
\hline
\end{array}
$$

3.3 Putting it all together

We can now combine the definitions in the preceding sections to specify the state space of the whole histogrammer hardware/software combination:

$$
\begin{array}{|l}
\hline
\textit{Histogrammer} \\
\textit{HistogrammerCard} \\
\textit{HistogramMemory} \\
\textit{RealMemory} \\
\textit{address} : \textit{Histogram} \times \textit{TUPLE} \to \mathbb{Z} \\
\textit{SpectrumStore} \\
\hline
\forall\, t : \textit{TAG} \bullet \textit{tagstate}\ t = \textit{enabled} \Rightarrow \\
\quad (\exists\, h : \textit{Histogram} \bullet \textit{membase}\ h = \textit{tagbase}\ t * 1024) \\
\forall\, s : \operatorname{ran} \textit{store} \bullet \textit{shapedimension}\ s.\textit{sshape} = 1 \land \textit{first}(\textit{head}\ s.\textit{sshape}) = 0 \\
\forall\, h : \operatorname{dom} \textit{membase};\ t : \textit{TUPLE} \mid t \operatorname{within} h.\textit{shape} \bullet \\
\quad \textit{address}(h, t) = \textit{RMbase} + \textit{membase}(h) + \textit{offset}(t, h.\textit{shape}) \land \\
\quad h.\textit{counts}(t) = \textit{memory}(\textit{address}(h, t)) \\
\hline
\end{array}
$$

Note the constraints in the *Histogrammer* schema: firstly, every enabled entry in the look-up table must point to a piece of memory associated with a histogram [1, section 3.6]; secondly, all spectra are one dimensional and start in channel zero; thirdly, the function $address(h, t)$ gives the address in histogramming memory of the channel t of histogram h; finally, we *identify* the count in channel t of histogram h with the contents of memory location $address(h, t)$.

Note also the implications of the histogrammer's use of bitor rather than addition. When presented with tag t and data d the histogrammer increments memory address

$$\textit{baseaddress} \operatorname{bitor} (\textit{tagbase}\ t * 1024) \operatorname{bitor} d.$$

We want this to be $address(h, c)$ where h is the histogram corresponding to tag t and $c = d$. Since all our histograms are one dimensional and start in channel zero this is just the address

$$\textit{RMbase} + \textit{membase}(h) + c.$$

To ensure that bitor gives the same result as addition we can insist that

$$\exists\, n \bullet \forall\, h \bullet \textit{RMbase} \bmod 2^n = 0 \land 2^n > \textit{membase}(h) + c$$

and that,

$$\forall\, h \bullet \exists\, m \bullet membase(h) \bmod 2^m = 0 \land 2^m > c.$$

These conditions are readily met by demanding that all histograms have power of two size and be aligned on their size and that *RMrange* also be a power of two with $RMbase \bmod RMrange = 0$.

$$\begin{array}{|l}
validranges : \mathbb{P}\,\mathbb{N} \\
\hline
validranges = \{1024, 2048, 4096, 8192\} \\
RMbase \bmod RMrange = 0
\end{array}$$

[Finally, note that $d = c$ and c within $h.shape$ implies d within $h.shape$ also. If the data presented to the histogrammer does not meet this condition, e.g., through some external hardware or programming fault, then spurious counts may appear in some spectra. It may be worthwhile widening the histogrammer's lookup table with per-tag mask fields to prevent this]

The next schema describes how the histogrammer system starts—basically nothing is known and all tags are disabled, but overall operation of the card is enabled. This avoids the need for explicit "halt" and "go" operations on the histogrammer hardware. It is not essential to zero the real memory at this point, though clearly some sort of memory self-test is desirable.

$$\begin{array}{|l}
\multicolumn{1}{l}{InitHistogrammer} \\
\hline
Histogrammer \\
\hline
state = enabled \\
\forall\, t : TAG \bullet tagstate\ t = disabled \\
membase = \varnothing \\
memrange = \varnothing \\
baseaddress = RMbase \\
store = \varnothing \\
\hline
\end{array}$$

In the foregoing we have tried to keep separate operations on abstract histograms from hardware descriptions. We now have to say how a histogram gets associated with a particular tag. First we reserve space for a histogram in the memory and then we *map* a certain tag onto that histogram.

$$\begin{array}{|l}
\multicolumn{1}{l}{MapTag} \\
\hline
\Delta HistogrammerCard \\
\Xi HistogramMemory \\
h? : Histogram \\
t? : TAG \\
\hline
state = disabled = state' \\
tagstate\ t? = disabled \\
tagstate' = tagstate \oplus \{t? \mapsto enabled\} \\
tagbase' = tagbase \oplus \{t? \mapsto membase\ h? \operatorname{div} 1024\} \\
baseaddress' = baseaddress \\
\hline
\end{array}$$

When we no longer require a histogram we must unmap the associated tag (if any) before we can free the memory belonging to the histogram.

$$
\begin{array}{|l}
\hline
\textit{UnMapTag} \\
\Delta \textit{HistogrammerCard} \\
\Xi \textit{HistogramMemory} \\
t? : \textit{TAG} \\
\hline
\textit{state} = \textit{disabled} = \textit{state}' \\
\textit{tagstate}\ t? = \textit{enabled} \\
\textit{tagstate}' = \textit{tagstate} \oplus \{t? \mapsto \textit{disabled}\} \\
\textit{tagbase}' = \{t?\} \mathbin{⩥} \textit{tagbase} \\
\textit{baseaddress}' = \textit{baseaddress} \\
\hline
\end{array}
$$

Both these operations require that the histogrammer hardware be in the disabled state, so let us give ourselves operations for enabling and disabling the histogrammer card.

$$
\begin{array}{|l}
\hline
\textit{Go} \\
\Delta \textit{HistogrammerCard} \\
\Xi \textit{HistogramMemory} \\
\hline
\textit{state} = \textit{enabled} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Stop} \\
\Delta \textit{HistogrammerCard} \\
\Xi \textit{HistogramMemory} \\
\hline
\textit{state}' = \textit{disabled} \\
\hline
\end{array}
$$

We can now promote these operations to the spectrum level:

$$
\begin{array}{l}
\textit{StopMapTagGo} \mathrel{\widehat=} \textit{Stop} \mathbin{⨾} \textit{MapTag} \mathbin{⨾} \textit{Go} \\
\textit{StopUnMapTagGo} \mathrel{\widehat=} \textit{Stop} \mathbin{⨾} \textit{UnMapTag} \mathbin{⨾} \textit{Go}
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{AssociateTag} \\
\textit{AccessSpectrum} \\
\textit{StopMapTagGo} \\
i? : \mathbb{N} \\
\hline
h? = s.\textit{arrays}\ i? \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{DissociateTag} \\
\textit{AccessSpectrum} \\
\textit{StopUnMapTagGo} \\
i? : \mathbb{N} \\
h : \textit{Histogram} \\
\hline
h = s.\textit{arrays}\ i? \\
t? = (\mu\, t : \textit{TAG} \mid \textit{tagbase}\ t * 1024 = \textit{membase}\ h) \\
\hline
\end{array}
$$

We can now completely define the operations of creating a new histogrammer spectrum and of deleting one:

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{CreateHistogrammerSpectrum}} \\
\Delta \mathit{Histogrammer} \\
\Xi \mathit{HistogrammerCard} \\
\mathit{CreateSpectrum} \\
m, r : \mathbb{Z} \\
\hline
\mathit{shapedimension}\ \mathit{spectrum.sshape} = 1 \wedge \mathit{first}(\mathit{head}\ \mathit{spectrum.sshape}) = 0 \\
r = \mathit{size}\ s? \wedge r \in \mathit{validranges} \\
m \bmod r = 0 \\
\mathrm{memfree}(m, r) \\
\mathit{membase}' = \mathit{membase} \oplus \{h \mapsto m\} \\
\mathit{memrange}' = \mathit{memrange} \oplus \{h \mapsto r\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{DeleteHistogrammerSpectrum}} \\
\Delta \mathit{Histogrammer} \\
t? : \mathit{TAG} \\
\mathit{DeleteSpectrum} \\
h : \mathit{Histogram} \\
\hline
h = (\mathit{store}\ \mathit{name}?).\mathit{arrays}\ 1 \\
(\exists\, t? : \mathit{TAG} \mid \mathit{tagbase}\ t? * 1024 = \mathit{membase}\ h \bullet \mathit{StopUnMapTagGo}) \vee \\
(\neg\, (\exists\, t? : \mathit{TAG} \mid \mathit{tagbase}\ t? * 1024 = \mathit{membase}\ h \bullet \Xi \mathit{HistogrammerCard})) \\
\mathit{membase}' = \{h\} \ntriangleright \mathit{membase} \\
\mathit{memrange} = \{h\} \ntriangleright \mathit{memrange} \\
\hline
\end{array}
$$

3.4 Error conditions

$$
\begin{array}{rcl}
\mathit{REPORT} & ::= & \mathit{ok} \\
& \mid & \mathit{domain\_outside\_spectrum} \\
& \mid & \mathit{invalid\_calibration\_number} \\
& \mid & \mathit{invalid\_annotation\_number} \\
& \mid & \mathit{invalid\_histogram\_number} \\
& \mid & \mathit{counts\_dont\_fill\_shape} \\
& \mid & \mathit{spectrum\_not\_known} \\
& \mid & \mathit{spectrum\_already\_exists} \\
& \mid & \mathit{shape\_unsuitable\_for\_histogrammer} \\
& \mid & \mathit{insufficient\_histogramming\_memory} \\
& \mid & \mathit{tag\_already\_being\_histogrammed} \\
& \mid & \mathit{histogram\_not\_associated\_with\_tag}
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{Ok}} \\
r! : \mathit{REPORT} \\
\hline
r! = \mathit{ok} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BadShape} \\
r! : REPORT \\
s? : SHAPE \\
h : Histogram \\
\hline
\neg\ rectdomain\ s? \subseteq rectdomain\ h.shape \\
r! = domain\_outside\_spectrum \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BadCalibration} \\
r! : REPORT \\
s : Spectrum \\
i? : \mathbb{N} \\
\hline
\neg\ i? \in 1 \mathrel{..} shapedimension\ s.sshape \\
r! = invalid\_calibration\_number \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BadAnnotation} \\
r! : REPORT \\
i? : \mathbb{N} \\
\hline
\neg\ i? \in 1 \mathrel{..} maxannotations \\
r! = invalid\_annotation\_number \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BadCounts} \\
r! : REPORT \\
i? : \mathbb{N} \\
s? : SHAPE \\
z? : \mathrm{seq}\,\mathbb{Z} \\
\hline
\#z? \neq size\ s? \\
r! = counts\_dont\_fill\_shape \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BadHistogram} \\
r! : REPORT \\
i? : \mathbb{N} \\
s : Spectrum \\
\hline
\neg\ i? \in \mathrm{dom}\, s.arrays \\
r! = invalid\_annotation\_number \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{UnknownSpectrum} \\
\Xi Histogrammer \\
r! : REPORT \\
n? : NAME \\
\hline
\neg\ n? \in \mathrm{dom}\, store \\
r! = spectrum\_not\_known \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AlreadyExists}} \\
\hline
\Xi \mathit{Histogrammer} \\
r! : \mathit{REPORT} \\
n? : \mathit{NAME} \\
\hline
n? \in \operatorname{dom} \mathit{store} \\
r! = \mathit{spectrum\_already\_exists} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{UnsuitableShape}} \\
\hline
s? : \mathit{SHAPE} \\
r! : \mathit{REPORT} \\
\hline
\neg\ (\mathit{shapedimension}\ \ s? = 1 \wedge \mathit{first}(\mathit{head}\ \ s?) = 0 \wedge \mathit{second}(\mathit{head}\ \ s?) \in \mathit{validranges}) \\
r! = \mathit{shape\_unsuitable\_for\_histogrammer} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NoMemory}} \\
\hline
\Xi \mathit{Histogrammer} \\
s? : \mathit{SHAPE} \\
r! : \mathit{REPORT} \\
\hline
\neg\ (\exists\, m, r : \mathbb{N} \bullet r = \mathit{size}\ \ s? \wedge m \bmod r = 0 \wedge \mathrm{memfree}(m, r)) \\
r! = \mathit{insufficient\_histogramming\_memory} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{TagInUse}} \\
\hline
\Xi \mathit{HistogrammerCard} \\
t? : \mathit{TAG} \\
r! : \mathit{REPORT} \\
\hline
\mathit{tagstate}\ \ t? = \mathit{enabled} \\
r! = \mathit{tag\_already\_being\_histogrammed} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NotAssociated}} \\
\hline
\Xi \mathit{HistogrammerCard} \\
\Xi \mathit{HistogramMemory} \\
t? : \mathit{TAG} \\
r! : \mathit{REPORT} \\
h : \mathit{Histogram} \\
\hline
\neg\ (\exists\, t : \mathit{TAG} \bullet \mathit{tagbase}\ \ t * 1024 = \mathit{membase}\ \ h) \\
r! = \mathit{histogram\_not\_associated\_with\_tag} \\
\hline
\end{array}
$$

3.5 Total operations

We can now specify total versions of our earlier operations:

$CreateSpectrumOp \mathrel{\widehat=} (CreateHistogrammerSpectrum \land Ok) \lor AlreadyExists$
$\quad \lor BadShape \lor NoMemory$
$DeleteSpectrumOp \mathrel{\widehat=} (DeleteHistogrammerSpectrum \land Ok) \lor UnknownSpectrum$
$ReadHistogramOp \mathrel{\widehat=} (ReadHistogram \land Ok) \lor UnknownSpectrum \lor BadHistogram$
$\quad \lor BadShape$
$WriteHistogramOp \mathrel{\widehat=} (WriteHistogram \land Ok) \lor UnknownSpectrum \lor BadHistogram \lor BadShape$
$\quad \lor BadCounts$
$ZeroHistogramOp \mathrel{\widehat=} (ZeroHistogram \land Ok) \lor UnknownSpectrum \lor BadHistogram \lor BadShape$
$\quad \lor BadCounts$
$ReadExtremaOp \mathrel{\widehat=} (ReadExtrema \land Ok) \lor UnknownSpectrum \lor BadHistogram$
$WriteExtremaOp \mathrel{\widehat=} (WriteExtrema \land Ok) \lor UnknownSpectrum \lor BadHistogram$
$ReadAnnotationOp \mathrel{\widehat=} (ReadAnnotation \land Ok) \lor UnknownSpectrum \lor BadAnnotation$
$WriteAnnotationOp \mathrel{\widehat=} (WriteAnnotation \land Ok) \lor UnknownSpectrum \lor BadAnnotation$
$ReadCalibrationOp \mathrel{\widehat=} (ReadCalibration \land Ok) \lor UnknownSpectrum \lor BadCalibration$
$WriteCalibrationOp \mathrel{\widehat=} (WriteCalibration \land Ok) \lor UnknownSpectrum \lor BadCalibration$
$AssociateTagOp \mathrel{\widehat=} (AssociateTag \land Ok) \lor UnknownSpectrum \lor BadHistogram \lor TagInUse$
$DissociateTagOp \mathrel{\widehat=} (DissociateTag \land Ok) \lor UnknownSpectrum \lor BadHistogram \lor NotAssociated$

References

[1] *Eurogam Project: Specification of Histogrammer*, John Alexander, October 1990, EDOC???.