

## APPENDIX A

### PROGRAMMING THE DSP TRANSIENT DIGITIZER/AVERAGING MEMORY

The DSP Technology, Inc. MODEL 2001AS is a 100MHz, 8-bit analog to digital converter. Connected to the 2001AS is the MODEL 4101 averaging memory. These units are powered by a CAMAC crate and are controlled through the CC-488 GPIB crate controller. The 2001AS and 4101 can only be controlled through the CC-488 via computer. Hence, successful operation of the DSP modules requires an in-depth understanding of the various CAMAC and GPIB codes.

When working with the DSP system, be aware: *THERE ARE MISTAKES IN THE MANUALS!* These various mistakes have impeded my progress many times, and whenever possible, I have noted the mistake. However, I have not used every aspect of the DSP system, so I cannot attest to the truth of every aspect of every manual. One very important error is the 2001AS – 4101 connector pinout. The correct pinout is located in table A-1.

2001AS Pin #	4101 Pin #	Function	Description
9	17	TRIG	Generator: 4101. Signals the 2001AS to begin digitizing.
10	19	EOC	Generator: 2001AS. Digitizing is complete and summation can begin. Remains high until RST is generated by the 4101. (20 ns pulse width)
22	18	MEMINC (or CNTADD)	Generator: 4101. Advances the memory of the 2001AS to the next location. (80 ns pulse width)
23	20	RST (or EOR)	Generator: 4101. Sweep is complete and 2001AS can gather more data. (80 ns pulse width)
17	8	BIT 7	Most significant bit from 2001AS.
4	7	BIT 6	--
16	6	BIT 5	--
3	5	BIT 4	--
15	4	BIT 3	--
2	3	BIT 2	--
14	2	BIT 1	--
1	1	BIT 0	Least significant bit from 2001AS.
25	--	CLK (1 MHz)	1 MHz clock from 2001AS
24	--	GND	2001AS ground

Table A-1. Pin assignments for the 2001AS – 4101 interface.

Controlling the DSP system requires a fairly good understanding of the GPIB interface. The particular GPIB card in use at the time of this writing is made by National Instruments. It is packaged with libraries for virtually every language and is very well documented. The code segments contained in this chapter are written in Microsoft Visual Basic v. 5.0. The computer used is a Pentium-75 running Windows 95. The DSP system is completely unaware of the specifics of the computer attached to the other end, so most of the information contained here will be applicable to any user of the DSP.

All communication between the computer and the various DSP components are done via the crate controller, in this case the CC-488 which can transfer 321 KBytes of data per second, as shown in figure A-1. In order to communicate to any of the units in the crate, you must send the appropriate codes to the crate controller. The crate controller then relays those codes to the appropriate device. The codes specific to a particular device are given in the respective instruction manual, but all the codes have a common form.

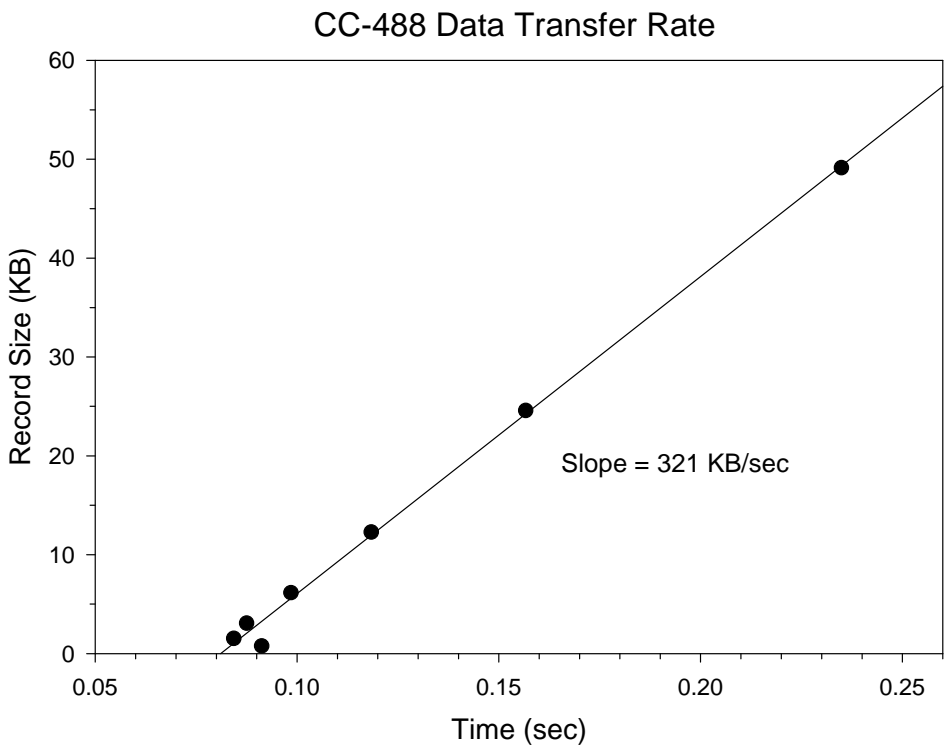


Figure A-1. The data transfer rate for the CC-488 CAMAC GPIB controller. The slope of the above line is 321 kBytes/sec.

**A.1 Reading Parameters from the DSP**

Actions performed on a DSP component fall into two categories: reading and writing. In order to read a parameter on a component requires three

components: (1) an 'N' code, (2) an 'A' code, and (3) an 'F' code. The 'N' code merely refers to the position of the particular component in the crate. For example, the 4101 averaging memory is in slot 24, so a read command sent to the 4101 must begin with 24. The 'F' code and 'A' code together refer to a specific function of the component. Tables of 'F' codes and 'A' codes are given in the manual. For example, to read the module ID for the 4101, you would need to enter A(0)F(3), or 0 in the 'A' code place and 3 in the 'F' code place. Once these codes are sent to the crate controller, the controller relays these codes to the proper module. The module then returns the requested value, which the controller intercepts and relays to the computer.

Communication between the DSP and the computer is done via 8-bit binary codes. These 8-bit binary codes are most commonly thought of as the ASCII character set, as shown in table A-2.

0	.	32	[space]	64	@	96	`	128	.	160	[space]	192	À	224	à
1	.	33	!	65	A	97	a	129	.	161	ı	193	Á	225	á
2	.	34	"	66	B	98	b	130	.	162	ø	194	Â	226	â
3	.	35	#	67	C	99	c	131	.	163	£	195	Ã	227	ã
4	.	36	\$	68	D	100	d	132	.	164	¤	196	Ä	228	ä
5	.	37	%	69	E	101	e	133	.	165	¥	197	Å	229	å
6	.	38	&	70	F	102	f	134	.	166	ı	198	Æ	230	æ
7	.	39	'	71	G	103	g	135	.	167	§	199	Ç	231	ç
8	**	40	(	72	H	104	h	136	.	168	"	200	È	232	è
9	**	41	)	73	I	105	i	137	.	169	©	201	É	233	é
10	**	42	*	74	J	106	j	138	.	170	ª	202	Ê	234	ê
11	.	43	+	75	K	107	k	139	.	171	«	203	Ë	235	ë
12	.	44	,	76	L	108	l	140	.	172	¬	204	Ì	236	ì
13	**	45	-	77	M	109	m	141	.	173	-	205	Í	237	í
14	.	46	.	78	N	110	n	142	.	174	®	206	Î	238	î
15	.	47	/	79	O	111	o	143	.	175	¯	207	Ï	239	ï
16	.	48	0	80	P	112	p	144	.	176	°	208	Ð	240	ð
17	.	49	1	81	Q	113	q	145	'	177	±	209	Ñ	241	ñ
18	.	50	2	82	R	114	r	146	'	178	²	210	Ò	242	ò
19	.	51	3	83	S	115	s	147	.	179	³	211	Ó	243	ó
20	.	52	4	84	T	116	t	148	.	180	´	212	Ô	244	ô
21	.	53	5	85	U	117	u	149	.	181	µ	213	Õ	245	õ
22	.	54	6	86	V	118	v	150	.	182	¶	214	Ö	246	ö
23	.	55	7	87	W	119	w	151	.	183	·	215	×	247	÷
24	.	56	8	88	X	120	x	152	.	184	¸	216	Ø	248	ø
25	.	57	9	89	Y	121	y	153	.	185	¹	217	Ù	249	ù
26	.	58	:	90	Z	122	z	154	.	186	º	218	Ú	250	ú
27	.	59	;	91	[	123	{	155	.	187	»	219	Û	251	û
28	.	60	<	92	\	124		156	.	188	¼	220	Ü	252	ü
29	.	61	=	93	]	125	}	157	.	189	½	221	Ý	253	ý
30	.	62	>	94	^	126	~	158	.	190	¾	222	Þ	254	þ
31	.	63	?	95	_	127	.	159	.	191	¿	223	ß	255	ÿ

Table A-2. The ASCII character set. The characters denoted by (·) do not have a visual representation. Asterisks (\*\*) represent control characters, such as the carriage return (ASCII code 13).



The `ibwrt()` and `ibrd()` functions are functions defined by the driver software for the particular GPIB card currently used in the lab. The functions take a string of ASCII characters and send them as their binary equivalents over the GPIB line. Obviously the number '4101' cannot be expressed by the DSP system as a single 8 bit number, so the crate controller breaks up the number into three 8 bit streams and sends them in the order of least significant byte to most significant byte. In order to read the result in a convenient way, the bytes must be reassembled to form the expected integer. If we were to examine each of the bytes from the DSP, we would see the following: 00000101 00010000 000 00000, which are the numbers 5, 16, and 0, respectively. Noting that the smallest byte is first, the result is given by:

$$\text{result} = 5 + (16 * 2^8) + (0 * 2^{16}) = 4096 + 5 = 4101,$$

which is the model number for the averaging memory.

## ***A.2 Writing Parameters to the DSP***

Writing a parameter to a DSP component is very similar to reading parameters. The write codes contain the familiar 'N', 'A', and 'F' codes, with the addition of another three-byte word known as the 'W' code. Again, all these codes must be converted to their ASCII-character equivalent before being sent. The following example demonstrates changing the number of averages on the 4101. The 4101 is programmed to stop summing when the number of sweeps reaches 65536. In order to obtain a number of averages less than 65536, you must set the initial value of the sweep register to 65536-n, where n is the desired number of averages, or sweeps. For the purposes of this example, the number of desired averages will be 5.

```

AverageNumber = 5
WCode = 65536-AverageNumber
Result = CrateWrite(24,0,17,Wcode)

If Result = 0 then
    Print "Success!"
Else
    Print "Failure!"
End if

Private Function CrateWrite(Ncode as long, Acode as long,
    Fcode as long, Wcode as long) As Integer

Dim Wdata3 As Integer
Dim Wdata2 As Integer
Dim Wdata1 As Integer

On Error GoTo ErrorHandler

WriteString$ = Space(3)
WriteString2$ = Space(3)

'Breaks the WCode into three bytes-----
Wdata3 = Int(WCode / 2 ^ 16)
Wdata2 = Int(WCode / 2 ^ 8) - Wdata3 * 2 ^ 8
Wdata1 = Int(WCode) - Wdata3 * 2 ^ 16 - Wdata2 * 2 ^ 8
'-----

NCode$ = Chr(NCode)           'Create Function Code to
ACode$ = Chr(ACode)           'send to DSP
FCode$ = Chr(FCode)

WriteString$ = Space(3)       'Allocate space for buffers
WriteString2$ = Space(3)

WriteString$ = NCode$ & ACode$ & FCode$           'Concatenate the
                                                    'strings
WriteString2$ = Chr(Wdata1) & Chr(Wdata2) & Chr(Wdata3)

Call ibwrt(DSPLocation%, WriteString$)           'Send the function
codes
Call ibwrt(DSPLocation%, WriteString2$)         'Send the Write code

CrateWrite = 0
Exit Function

ErrorHandler:
CrateWrite = -1

End Function

```

It is important to note the size of the integers being written to the DSP. In this case, the integer size is 65531, which is a number too large for the integer data type to hold. Therefore, it is necessary to work with 4-byte integers or the long data

type (see table A-3). In addition, it is again important to break up the long integer into a 3-byte word in the order of least significant byte to most significant byte.

Data Type	Size	Range
Byte	1 byte	0 to 255
Boolean	2 bytes	True or False
Integer	2 bytes	-32,768 to 32,767
Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision floating-point)	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.00000000000000000000000000000001
Date	8 bytes	January 1, 100 to December 31, 9999
Object	4 bytes	Any Object reference
String (variable-length)	10 bytes + string length	0 to approximately 2 billion

Table A-3. Data Types and sizes.

### **A.3 Block Transfer**

There is a special mode for the transfer of entire traces. The block transfer mode is very similar to the read mode, but instead of sending a single 3 byte word, the controller sends a stream of 3 byte words representing every point in the trace. These streams are intercepted by the GPIB software and stored as a string. The largest trace that can be stored by the digitizer/memory combination is 32768 data points (32K). Each point is represented by a 3 byte word which results in a string that is at most 98304 bytes long. In order to read the data, the software must loop through the string and combine every three bytes into a single long integer and store that integer in an array. Each element in that array must then be divided by the number of sweeps made by the averaging memory in order to yield the true average of the data. The following code demonstrates initiating the block transfer command to the 4101 and interpreting the results.

```
Private Sub BlockTransfer(NCode as long, FCode as long,
                        ACode as long, RecLength as long, AverageNum as long)

Dim Length As Long
Dim result(1 To 3) As Long

Buffer$ = Space(3 * RecLength)      'Allocate enough space
WriteString$ = Space(3)             'for buffers

NCode$ = Chr(NCode)                 'create ASCII character
ACode$ = Chr(ACode)                 'representations
FCode$ = Chr(FCode)
WriteString$ = NCode$ & ACode$ & FCode$

Call ibwrt(DSPLocation%, WriteString$) 'Send the command

Call ibrd(DSPLocation%, Buffer$)      'Receive the data

Length = RecLength * 3
ReDim Data(RecLength)                'Dimension the data array
```

```

k = 0
For i = 1 To Length Step 3

    result(1) = Asc(Mid(Buffer$, i, 1))
    result(2) = Asc(Mid(Buffer$, i + 1, 1))
    result(3) = Asc(Mid(Buffer$, i + 2, 1))

    `Convert the array to integers-----
        Data1(k) = result(1) + (result(2) * 256) +
            (result(3) * 65536)
    `-----

    `Convert the integers to Volts-----
        Data1(k) = ((Data1(k) / AverageNum) - 129) / 493
    `-----

    k = k + 1
Next i

End Sub

```

#### ***A.4 Setting up the Digitizer/Averaging Memory***

The power on state of the DSP system is not suitable for taking data. There are a few key variables that must be initialized. A system initialization flow chart is contained in figure A-2. Setting up the 2001AS digitizer requires sending a single 11-bit integer. The value of this integer determines the number of pretrigger samples, record length, and sampling interval. Table A-4 contains the correct bit positions for each parameter. It is important to make sure the record length for the 4101 and the 2001AS are set to the same value.

Parameter	Value	Bit Position										
		11	10	9	8	7	6	5	4	3	2	1
Pre-Trig Samples	0	--	--	--	--	--	--	--	--	0	0	0
	1/8	--	--	--	--	--	--	--	--	0	0	1
	2/8	--	--	--	--	--	--	--	--	0	1	0
	3/8	--	--	--	--	--	--	--	--	0	1	1
	4/8	--	--	--	--	--	--	--	--	1	0	0
	5/8	--	--	--	--	--	--	--	--	1	0	1
	6/8	--	--	--	--	--	--	--	--	1	1	0
	7/8	--	--	--	--	--	--	--	--	1	1	1
Record Length	256	--	--	--	--	--	1	0	1	--	--	--
	512	--	--	--	--	--	1	0	0	--	--	--
	1024	--	--	--	--	--	0	1	1	--	--	--
	2048	--	--	--	--	--	0	1	0	--	--	--
	4096	--	--	--	--	--	0	0	1	--	--	--
	8192	--	--	--	--	--	0	0	0	--	--	--
	16384	--	--	--	--	--	1	1	0	--	--	--
	32768	--	--	--	--	--	1	1	1	--	--	--
Sampling Interval	10ns	--	--	0	0	0	--	--	--	--	--	--
	20ns	--	--	0	0	1	--	--	--	--	--	--
	50ns	--	--	0	1	0	--	--	--	--	--	--
	100ns	--	--	0	1	1	--	--	--	--	--	--
	200ns	--	--	1	0	0	--	--	--	--	--	--
	500ns	--	--	1	0	1	--	--	--	--	--	--
	1000ns	--	--	1	1	0	--	--	--	--	--	--
	Ext. Clk.	--	--	1	1	1	--	--	--	--	--	--
Local		1	--	--	--	--	--	--	--	--	--	--
Remote		0	--	--	--	--	--	--	--	--	--	--

Table A-4. Bit positions for the setup parameters for the 2001AS.

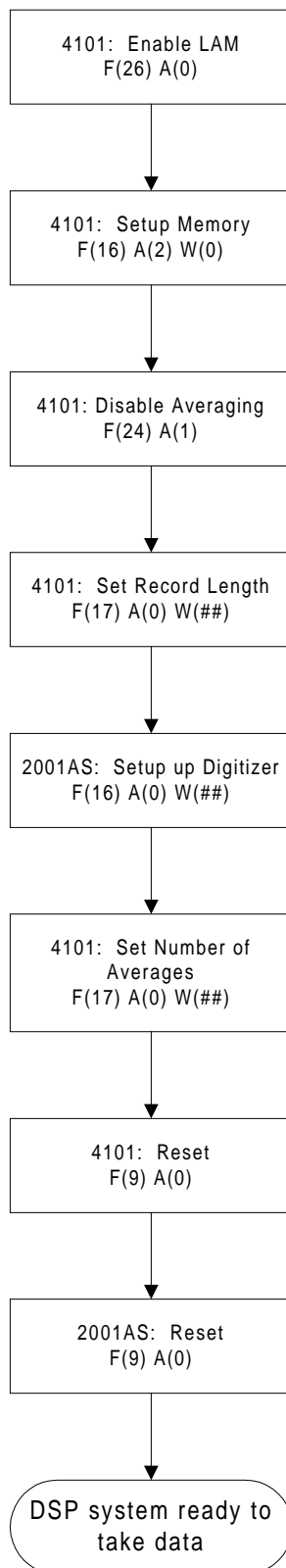


Figure A-2. Flow chart outlining the procedure for initializing the 4101/2001AS

### ***A.5 Using the Digitizer/Averaging Memory***

Using the 2001AS/4101 combination is fairly straight forward, but there are a few caveats. First of all, the signal input for the 2001AS has a 50 ohm input impedance. Also, the input dynamic range for the 2001AS is 0.5 volts. There is included on the 2001AS a trimpot labeled “Offset Adjust” which allows the user to move the 0.5 volt input range about 0.0 volts. Currently, the offset adjust is set to 0.25 volts, which allows the 2001AS to accept voltages between –0.25 volts and 0.25 volts. Any voltages outside the range of the 2001AS are clipped.

The 2001AS is an 8-bit digitizer which means the input voltage can take any integer value between 0 and 255. Converting these values back into voltage requires knowledge of two parameters: the offset and the voltage step size. The offset is easily changed on the front panel of the 2001AS, but the step size is built into the analog to digital circuitry of the 2001AS. Currently, the offset adjust is set to 0.25 volts or a digital value of 129. This implies that a grounded input will output a digital value of 129; a value of –0.25 volts will yield a digital value of 0; and a value of 0.25 volts will yield a digital value of 255. A plot of voltage vs digital value for a number of points is contained in figure A-3. The numbers from this plot have been included in the software to convert the digital value to a voltage value using the following equation:

$$\text{Voltage} = \frac{\text{digital\_number} - 129}{493 \text{ volts}^{-1}}.$$

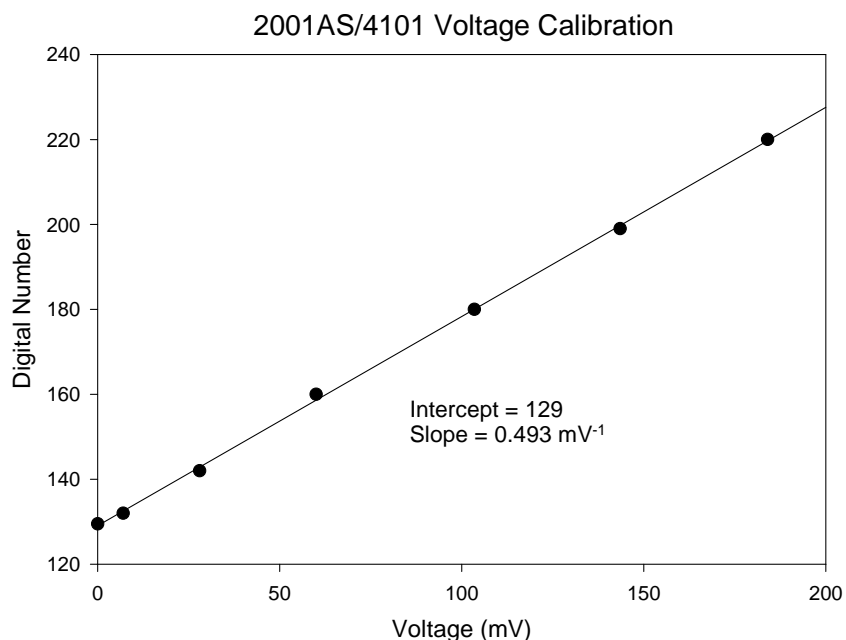


Figure A-3. Voltage calibration plot for the 4101. The above line has an intercept of 129 and a slope of 493 volts<sup>-1</sup>.

Be aware that the 4101 does not divide the values in its memory by the number of sweeps. In that respect, it is not a rigorous averager, but instead only a summer. It is up to the user to divide the result by the number of sweeps acquired.

### **A.6 The Scanner 1.0 Program**

Scanner 1.0 is the first program written to take advantage of the abilities of the 2001AS/4101 combination. A line by line explanation of the code is beyond the scope of this thesis, but a brief overview of the algorithms and equations involved is appropriate. The program is designed to be a graphical, user-friendly gated integrator and frequency controller to the F-Center laser in B214. The software uses the DSP system for the acquisition of the transient absorbance signals and uses an

inexpensive National Instruments A-D/D-A card for the acquisition of various other DC voltages and laser frequency control. The software displays the absorbance signal, the TAC signal from a scanning fabry-perot etalon, and the IR DC power . The program uses a gated integration technique for gathering data at each frequency. The user is able to choose the position and width of two gates, one that includes baseline and the other that includes the transient absorbance signal. The software sums the points in each gate and calculates the difference. That difference can be compared to the IR DC level to yield an absorbance value. A flow chart containing the algorithm for the primary data taking loop is contained in figure A-4.

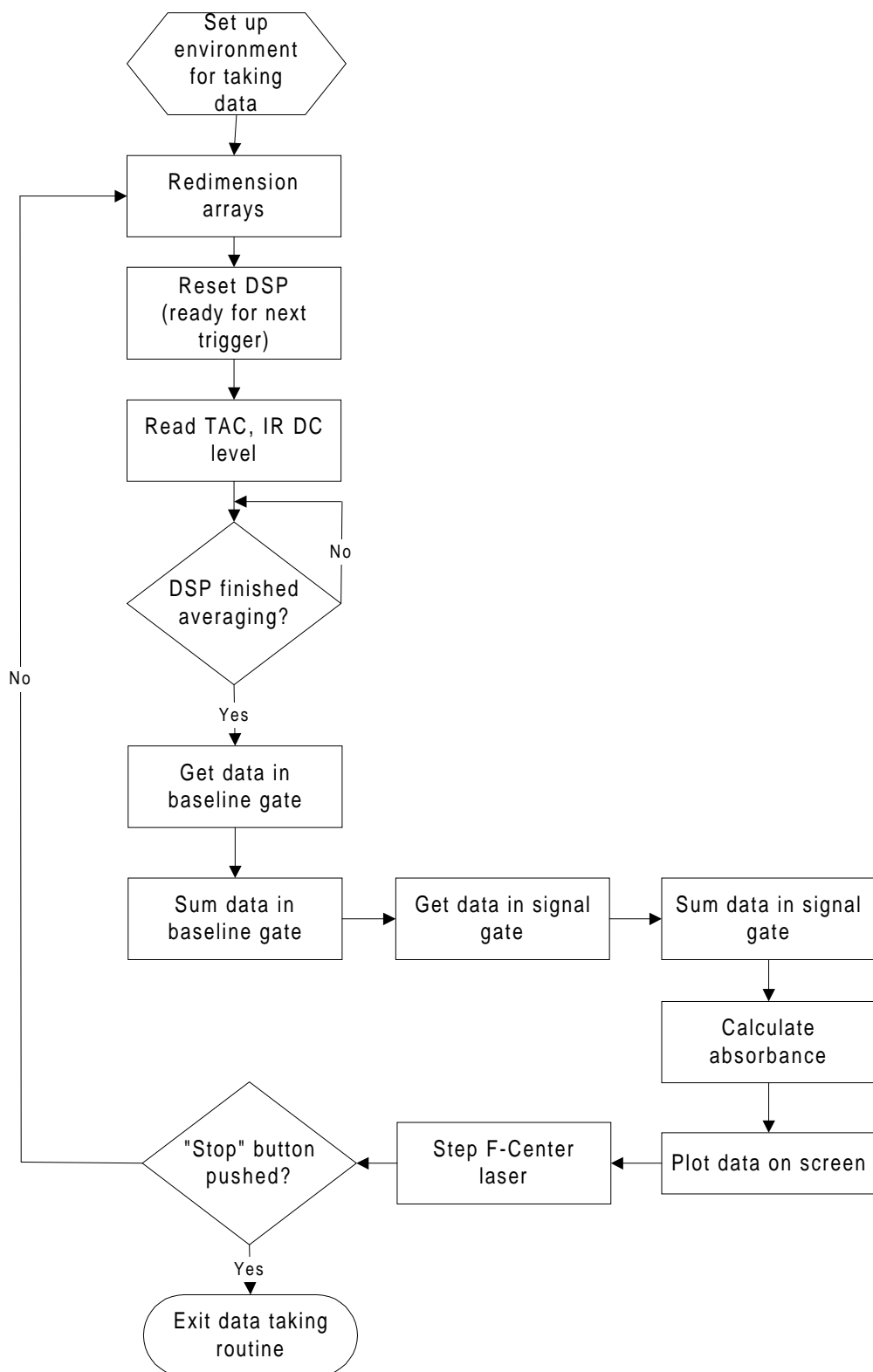


Figure A-4. Flow chart outlining the data taking subroutine in Scanner 1.0.

There are a few notable features about the loop contained in figure A-4.

First of all, only the data within the gates is transferred from the DSP. This reduces the amount of data transferred and increases the speed at which data can be taken.

Secondly, the equation for calculating the absorbance is as follows:

$$A = -\ln \left( -\frac{\text{SignalSum} - \text{BaseSum}}{\text{GateWidth} \times \text{Gain} \times \text{DCLevel}} + 1 \right),$$

where Gain is the total gain of the subtracted signal and DCLevel is the amount of IR light (in volts) on the reference detector. This equation is valid as long as the signal gate and the baseline gate are of the same size, which is a requirement in this particular version of Scanner.

Scanner 1.0 saves the data and parameters to a binary file. This saves disk space and file access time. However, working with binary files requires forehand knowledge of the order and type of variables in the file. The following is a section of the subroutine that saves the binary file.

```
Open FileName For Binary As #3
Put #3, , Index           `Number of data points (integer)
Put #3, , PreTrig        `DSP pre-trigger samples (integer)
Put #3, , SampInterval   `DSP sampling interval (integer)
Put #3, , RecLength      `DSP record length (long integer)

Put #3, , Data           `Data structure that holds all data
Close #3
```

The data structure Data has the following format:

```
Type DataRecord
SignalData() As Single   `Subtracted signal (Volts)
Absorbance() As Single   `Calculated Absorbance
IRData() As Single       `IR DC level (Volts)
TACdata() As Single      `TAC data (Volts)
Average As Integer       `Number of averages
Index As Long            `Number of data points
FrequencyStep As Integer `F-Center Step size
Gain() As Single         `Reported Gain at each point
SigGate1 As Long         `Index of start of signal gate
```

```

SigGate2 As Long           'Index of end of signal gate
BaseGate1 As Long         'Index of start of baseline gate
basegate2 As Long        'Index of end of baseline gate
GateWidth As Long        'Width of signal and baseline gate
XScale As Long           'Scanner 1.0 horizontal zoom factor
Frequency() As Single     'Frequency of each point
FrequencyIndex As Long    'Number of frequency markers in scan
FrequencyMarker() As Long 'Location of frequency markers
End Type

```

The sizes of each data type are given in table A-3.

In order to read the binary file into another program, the programmer must make a compatible data structure to the one shown above. The actual data names are irrelevant, but the sizes of each variable and array are crucial. Next, one must open the proper file and read the first four integers (the last of which is a long integer). Now, the file can be read into the compatible structure. The following is an example of a routine that correctly reads a Scanner 1.0 file:

```

'Scanner 1.0 Type Definition-----
Type Scanner1Data
  SignalData() As Single
  Absorbance() As Single
  IRData() As Single
  TACdata() As Single
  Average As Integer
  Index As Long
  FrequencyStep As Integer
  Gain() As Single
  SigGate1 As Long
  SigGate2 As Long
  BaseGate1 As Long
  BaseGate2 As Long
  GateWidth As Long
  XScale As Long
  Frequency() As Single
  FrequencyIndex As Long
  FrequencyMarker() As Long
End Type
'-----

'Dimension Some Local Variables----
Dim Scn1 As Scanner1Data

```

```
Dim PreTrig As Integer
Dim SampInterval As Integer
Dim RecLength As Long
'-----

'Open file for read-only,binary acces-----
Open FileName For Binary Access Read As #1
'-----

'Get some initial data parameters-----
Get #1, , Index
Get #1, , PreTrig
Get #1, , SampInterval
Get #1, , RecLength
'-----

'Get Majority of data-----
Get #1, , Scn1
'-----

'Close File-----
Close #1
'-----
```