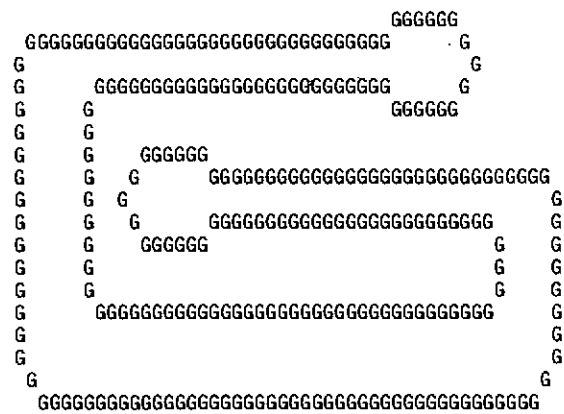
SUPERVISOR SBD User Manual

VOLUME 2 - Chapters 4-9

Technical Manual

Issue: 2.1 (December 1985)
Releases: A 2.1, B 2.1
Part No.: 4055022



GRESHAM LION (PPL) LTD LOWER WAY THATCHAM BERKSHIRE RG13 4RE U.K.
A DOWTY Group Company

The information in this document is subject to change without notice and should not be construed as a commitment by Gresham Lion (PPL) Ltd. Gresham Lion (PPL) Ltd assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished for use on a single Supervisor SBD installation and may not be used on other installations or for other purposes without the express permission of Gresham Lion (PPL) Ltd.

No responsibility is assumed for the use of such software on equipment not supplied by Gresham Lion (PPL) Ltd, or for the use of software not supplied by Gresham Lion (PPL) Ltd on any Supervisor installation.

The following are trademarks of Gresham Lion (PPL) Ltd.:

Supervisor 214	Supervisor 1024	Supervisor SBD
Supervisor PCU	Arcaid	GGS

The following are trademarks of Digital Equipment Corp.:

Q-bus	UNIBUS	RT-11
RSX	VAX	DEC
LSI-11	VMS	VT100
MicroVAX		

UNIX is a trademark of AT & T Bell Labs.
ID and Bit Pad One are trademarks of Summagraphics Corp.

C O N T E N T S - Volume 2

CHAPTER 4 - CONTENTS

Figures

Tables

4 Use of Peripheral Devices

A flexible system has been devised to allow the keyboard serial port on the SBD to be used for control of one or more of a number peripheral devices. The peripherals which are currently available for use with the SBD are the following:

Keyboard, with/without PCU (Key Tronic)
Trackerball (Marconi or Penny & Giles)
Graphics Tablet (TDS or Summagraphics Bit Pad One)
Digitising Table (Summagraphics ID)
Mouse (Penny & Giles)
Inkjet Printer (Integrex)
Frame Buffer/Hardcopier (Graftel/Canon)

If more than one peripheral is required then the keyboard can be supplied with a Peripheral Control Unit (PCU) included inside its case. The remaining peripherals are then plugged into the rear of the keyboard and the keyboard is plugged into the SBD - see figures 2.6 and 2.5.

When the SBD powers up, is reset by an initialisation of the host bus, or exits from SETUP, it checks a number of connections made at the keyboard connector. The state of these indicates which peripheral is connected. If a PCU is connected, it tells the SBD which peripherals are connected to it via a communication protocol and uses the connections to describe the readiness of the printer. (No printer is reported as printer not ready.)

To achieve the coding described above there are two inputs to the SBD (S0 and S1) and one output (CON). Table 4.1 describes the coding. Pin numbers are on the 10-pin AMP connector (see Figure 2.5, section 2.2.3). S0 and S1 are high if not connected and may be pulled low either by the peripheral concerned (e.g. the printer) or by connection within the cable to 0 volts.

Cables assembled with the correct coding can be supplied by Gresham Lion (PPL) Ltd. Part numbers of the cables for each peripheral are given in chapter 1.9.

S0 (Pin 1)	S1 (Pin 5)	Meaning
High	High	Nothing connected
High	Low	Keyboard / external VDU
High	Con	(1) ISI
Low	High	Printer - Not Ready
Low	Low	Printer - Ready
Low	Con	Tablet
Con	High	(2) PCU - Printer not ready
Con	Low	(2) PCU - Printer ready
Con	Con	Not used

Con is Pin 4. All levels are TTL.

Table 4.1 Peripheral Decode Pins on SBD Keyboard Socket

Note (1). The ISI (Intelligent Serial Interface) must be used when connecting a trackerball or mouse directly to the SBD (i.e. a PCU is not used).

Note (2). Other peripherals apart from the keyboard (tablet/table, trackerball/mouse, printer) may also be attached via the PCU. The SBD cannot sense this directly, but obtains the information by negotiation with the PCU (see 4.2.1).

4.1 Keyboard

The keyboard is an option which is normally required when the Supervisor SBD is to function as a VDU, or when more than one peripheral device needs to be attached to the SBD's peripheral port simultaneously.

Two types of keyboard are available. One simply provides the VDU capability and does not allow the attachment of other peripherals simultaneously. The other type, known as the Peripheral Controller Unit (PCU) houses three additional sockets which can be used to attach such devices as a trackerball, mouse, printer or tablet. The PCU is described in greater detail in Chapter 4.2.

The keyboard is a DEC VT100 layout low profile design which is shown in Figure 4.1. The cable from the SBD provides power as well as RS 423 and control lines. When power is applied to the SBD, the On-Line LED on the keyboard will light and its buzzer will sound briefly. The four LED's, L1-L4, are user programmable from the host.

The SBD should be connected to the host computer via the host serial port (see Figure 2.5). The SBD, keyboard and monitor can now be used as a VDU.

With the exceptions listed below, all key depressions result in the transmission of a code or codes to the host via the host serial line according to the currently set host line characteristics. Raw codes sent to the SBD are listed in Table 4.2. For further information, see also Chapter 9.

- VDU PIX alternately selects the VDU data and pixel data to appear alone on the screen. When in MIX mode (see below) it causes the VDU data to be selected initially.

- MIX selects mixed pixel/VDU data to appear on the screen (split-screen, not overlay). The VDU lines are in the lower part of the screen. The number of VDU lines displayed depends on the value currently selected in SETUP (see 2.3) or the argument to the last MIX command received (see the PIX, VDU and MIX commands in 3.10)

- Direction Keys control the software pixel cursor when it is activated by the CUR or TRA commands. Otherwise they transmit the normal escape sequences to the host.

- BREAK, when pressed together with the CTRL and SHIFT keys, causes the monitor to enter SETUP (see 2.3). When BREAK is pressed alone or with the SHIFT key, a break condition is forced on the host line. When pressed with the CTRL key, the answerback message is sent to the host (see 9.1.6).

- CAPS LOCK, CONTROL and SHIFT Keys do not transmit when pressed alone. When the red LED on the CAPS LOCK key is lit, it indicates that alphabetic characters will be sent to the host as upper case.

- Keys on the keypad transmit the normal VT100 characters or escape sequences according to the mode in operation. When the graphics cursor is active, these keys act as terminators - see 3.10 (CUR and TRA).

- SETUP key enters Terminal Setup mode which allows certain VT100 characteristics to be defined (see 9.5).

- If the SBD is unable to transmit to the host for any reason (e.g. XOFF received, or characters coming from the keyboard faster than they can be sent to the host) then the SBD internally buffers characters to be sent (some keys generate more than one character in this buffer). The capacity of the buffer is 16 characters. When the buffer is full, the SBD lights the "KBD LOCKED" LED on the keyboard and sounds the bell.

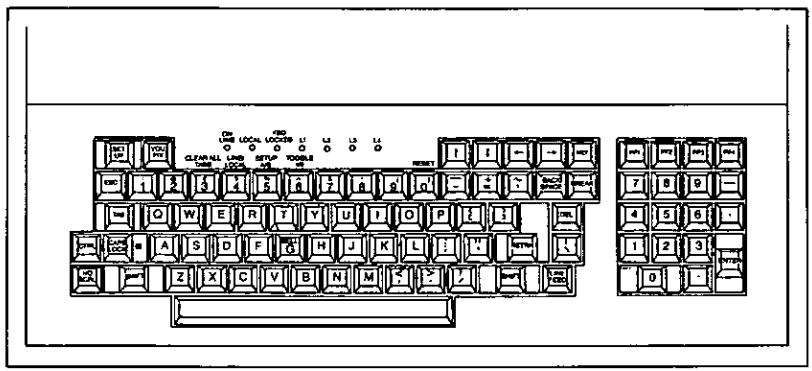


Figure 4.1 Keyboard Layout

4.1.1 Keyboard Codes

Table 4.2 lists the raw 8-bit codes which are produced by the SBD keyboard (simple, PCU or Cherry) and sent to the SBD. If any other keyboard is interfaced to the SBD it must also produce these codes. Note: there are 6 blank keys on the Cherry keyboard. These are numbered in raster fashion (Blank1 = key next to F10, Blank6 = key under MENU).

Table 4.2 Raw codes from keyboard to SBD

Key: Type = M : main keyboard) Symbol shown is key engraving,
= P : keypad) or lower of the two engravings
= O : other peripheral
= C : available on Cherry keyboard only
= N : available on standard keyboard only

Mode = S : shift
C : control
SC: control plus shift
A : all (independent of control and shift)
U : unshift (key alone)

Code = decimal value (plus ASCII symbol/name if any) or escape sequence sent by SBD to host computer when key pressed. Refer to Chapter 9 for more details.

--- Code ----			Key	Type	Mode	Code (decimal) or escape
Oct	Dec	Hex				sequence sent to host
0	0	0	2	M,N	C,CS	0 (NUL)
			SPACE	M,C	C,CS	0 (NUL)
1	1	1	A	M	C,CS	1 (SOH)
2	2	2	B	M	C,CS	2 (STX)
3	3	3	C	M	C,CS	3 (ETX)
4	4	4	D	M	C,CS	4 (EOT)
5	5	5	E	M	C,CS	5 (ENQ)
6	6	6	F	M	C,CS	6 (ACK)
7	7	7	G	M	C,CS	7 (BEL)
10	8	8	H	M	C,CS	8 (BS)
			BACK SPACE	M	A	8 (BS)
11	9	9	I	M	C,CS	9 (HT)
			TAB	M	A	9 (HT)
12	10	A	J	M	C,CS	10 (LF)
			LINE FEED	M	A	10 (LF)
13	11	B	K	M	C,CS	11 (VT)
14	12	C	L	M	C,CS	12 (FF)
15	13	D	M	M	C,CS	13 (CR)
			RETURN	M	A	13 (CR) or 13,10 (CR)(LF)

Table 4.2 continued

Code Oct	Code Dec	Code Hex	Key	Type	Mode	Code (decimal) or escape sequence sent to host
16	14	E	N	M	C,CS	14 (SO)
17	15	F	O	M	C,CS	15 (SI)
20	16	10	P	M	C,CS	16 (DLE)
21	17	11	Q	M	C,CS	17 (DC1) or (XON)
22	18	12	R	M	C,CS	18 (DC2)
23	19	13	S	M	C,CS	19 (DC3) or (XOFF)
24	20	14	T	M	C,CS	20 (DC4)
25	21	15	U	M	C,CS	21 (NAK)
			REJ	P,C	A	21 (NAK)
26	22	16	V	M	C,CS	22 (SYN)
27	23	17	W	M	C,CS	23 (ETB)
30	24	18	X	M	C,CS	24 (CAN)
31	25	19	Y	M	C,CS	25 (EM)
32	26	1A	Z	M	C,CS	26 (SUB)
33	27	1B	[	M	C,CS	27 (ESC)
			ESC	M	A	27 (ESC)
34	28	1C	\	M	C,CS	28 (FS)
35	29	1D	]	M	C,CS	29 (GS)
36	30	1E	6	M,N	C,CS	30 (RS)
			^	M,C	C,CS	30 (RS)
37	31	1F	-	M,N	C,CS	31 (US)
			/	M,C	C,CS	31 (US)
40	32	20	SPACE	M	A	32 (SP)
41	33	21	1	M	S,CS	33 (!)
42	34	22	'	M	S,CS	34 (")
43	35	23	3	M	C	35 (# or hash)
44	36	24	4	M	S,CS	36 ($)
45	37	25	5	M	S,CS	37 (%)
46	38	26	7	M	S,CS	38 (&)
47	39	27	'	M	U,C	39 (')
50	40	28	9	M	S,CS	40 (()
51	41	29	0	M	S,CS	41 ())
52	42	2A	8	M	S,CS	42 (*)
53	43	2B	=	M	S,CS	43 (+)
54	44	2C	,	M	U,C	44 (,)
55	45	2D	-	M	U	45 (-)
56	46	2E	.	M	U,C	46 (.)
57	47	2F	/	M	U,C	47 (/)
60	48	30	0	M	U,C	48 (0)
61	49	31	1	M	U,C	49 (1)
62	50	32	2	M	U	50 (2)
63	51	33	3	M	U	51 (3)
64	52	34	4	M	U,C	52 (4)
65	53	35	5	M	U,C	53 (5)
66	54	36	6	M	U	54 (6)

Table 4.2 continued

Code Oct	Code Dec	Code Hex	Key	Type	Mode	Code (decimal) or escape sequence sent to host
67	55	37	7	M	U,C	55 (7)
70	56	38	8	M	U,C	56 (8)
71	57	39	9	M	U,C	57 (9)
72	58	3A	:	M	S,CS	58 (:)
73	59	3B	;	M	U,C	59 (;)
74	60	3C	,	M	S,CS	60 (<)
75	61	3D	=	M	U,C	61 (=)
76	62	3E	.	M	S,CS	62 (>)
77	63	3F	/	M	S,CS	63 (?)
100	64	40	2	M	S	64 (@)
101	65	41	A	M	S	65 (A)
102	66	42	B	M	S	66 (B)
103	67	43	C	M	S	67 (C)
104	68	44	D	M	S	68 (D)
105	69	45	E	M	S	69 (E)
106	70	46	F	M	S	70 (F)
107	71	47	G	M	S	71 (G)
110	72	48	H	M	S	72 (H)
111	73	49	I	M	S	73 (I)
112	74	4A	J	M	S	74 (J)
113	75	4B	K	M	S	75 (K)
114	76	4C	L	M	S	76 (L)
115	77	4D	M	M	S	77 (M)
116	78	4E	N	M	S	78 (N)
117	79	4F	O	M	S	79 (O)
120	80	50	P	M	S	80 (P)
121	81	51	Q	M	S	81 (Q)
122	82	52	R	M	S	82 (R)
123	83	53	S	M	S	83 (S)
124	84	54	T	M	S	84 (T)
125	85	55	U	M	S	85 (U)
126	86	56	V	M	S	86 (V)
127	87	57	W	M	S	87 (W)
130	88	58	X	M	S	88 (X)
131	89	59	Y	M	S	89 (Y)
132	90	5A	Z	M	S	90 (Z)
133	91	5B	[	M	U	91 ([)
134	92	5C	\	M	U	92 (\)
135	93	5D	]	M	U	93 (])
136	94	5E	6	M	S	94 (^)
137	95	5F	-	M	S	95 (_) (underscore)
140	96	60	`	M,N	U,C	96 (`)
			`	M,C	U	96 (`)
141	97	61	A	M	U	97 (a)
142	98	62	B	M	U	98 (b)

Table 4.2 continued

Code Oct	Code Dec	Code Hex	Key	Type	Mode	Code (decimal) or escape sequence sent to host
143	99	63	C	M	U	99 (c)
144	100	64	D	M	U	100 (d)
145	101	65	E	M	U	101 (e)
146	102	66	F	M	U	102 (f)
147	103	67	G	M	U	103 (g)
150	104	68	H	M	U	104 (h)
151	105	69	I	M	U	105 (i)
152	106	6A	J	M	U	106 (j)
153	107	6B	K	M	U	107 (k)
154	108	6C	L	M	U	108 (l)
155	109	6D	M	M	U	109 (m)
156	110	6E	N	M	U	110 (n)
157	111	6F	O	M	U	111 (o)
160	112	70	P	M	U	112 (p)
161	113	71	Q	M	U	113 (q)
162	114	72	R	M	U	114 (r)
163	115	73	S	M	U	115 (s)
164	116	74	T	M	U	116 (t)
165	117	75	U	M	U	117 (u)
166	118	76	V	M	U	118 (v)
167	119	77	W	M	U	119 (w)
170	120	78	X	M	U	120 (x)
171	121	79	Y	M	U	121 (y)
172	122	7A	Z	M	U	122 (z)
173	123	7B	[	M	S	123 ({)
174	124	7C	\	M	S	124 (|)
175	125	7D	]	M	S	125 (})
176	126	7E	~	M,N	S,CS	126 (~)
			`	M,C	S	126 (~)
177	127	7F	DELETE	M	A	127 (DEL)
			DEL	P,C	A	127 (DEL)
200	128	80	Tball/Mouse	0	-	None
201	129	81	Tball/Mouse	0	-	None
202	130	82	Tball/Mouse	0	-	None
203	131	83	Tball/Mouse	0	-	None
204	132	84	Tball/Mouse	0	-	None
205	133	85	Tball/Mouse	0	-	None
206	134	86	Tball/Mouse	0	-	None
207	135	87	Tball/Mouse	0	-	None
210	136	88	Tball/Mouse	0	-	None
211	137	89	Tball/Mouse	0	-	None
212	138	8A	Tball/Mouse	0	-	None
213	139	8B	Tball/Mouse	0	-	None
214	140	8C	Tball/Mouse	0	-	None
215	141	8D	Tball/Mouse	0	-	None

Table 4.2 continued

Code Oct	Code Dec	Code Hex	Key	Type	Mode	Code (decimal) or escape sequence sent to host
216	142	8E	Tball/Mouse	0	-	None
217	143	8F	Tball/Mouse	0	-	None
220	144	90	Tball/Mouse	0	-	None
221	145	91	Tball/Mouse	0	-	None
222	146	92	Tball/Mouse	0	-	None
223	147	93	Tball/Mouse	0	-	None
224	148	94	Tball/Mouse	0	-	None
225	149	95	Tball/Mouse	0	-	None
226	150	96	Tball/Mouse	0	-	None
227	151	97	Tball/Mouse	0	-	None
230	152	98	Tball/Mouse	0	-	None
231	153	99	Tball/Mouse	0	-	None
232	154	9A	Tball/Mouse	0	-	None
233	155	9B	Tball/Mouse	0	-	None
234	156	9C	Tball/Mouse	0	-	None
235	157	9D	Tball/Mouse	0	-	None
236	158	9E	Tball/Mouse	0	-	None
237	159	9F	Tball/Mouse	0	-	None
240	160	A0				
241	161	A1	PIX	C	A	None (screen control)
242	162	A2	MIX	C	A	None (screen control)
243	163	A3				
244	164	A4				
245	165	A5				
246	166	A6				
247	167	A7				
250	168	A8				
251	169	A9				
252	170	AA				
253	171	AB				
254	172	AC				
255	173	AD				
256	174	AE				
257	175	AF				
260	176	B0	NO SCROLL	M	A	19 (DC3) or 17 (DC1)
261	177	B1	0	P	A	48 (0) or <ESC>Op
262	178	B2	.	P	A	46 (.) or <ESC>On
263	179	B3	Tball/Mouse	0	-	None
264	180	B4				
265	181	B5				
266	182	B6				
267	183	B7				
270	184	B8				
271	185	B9				
272	186	BA				

Table 4.2 continued

Code Oct	Code Dec	Code Hex	Key	Type	Mode	Code (decimal) or escape sequence sent to host
273	187	BB				
274	188	BC				
275	189	BD				
276	190	BE				
277	191	BF				
300	192	C0	1	P	A	49 (1) or <ESC>Oq
			Tball/Mouse	O	-	None
301	193	C1	2	P	A	50 (2) or <ESC>Or
			Tball/Mouse	O	-	None
302	194	C2	3	P	A	51 (3) or <ESC>Os
			Tball/Mouse	O	-	None
303	195	C3	ENTER	P	A	13 (CR) or <ESC>OM
			INJ	P,C	A	13 (CR) or <ESC>OM
304	196	C4	F1	C	A	<ESC>!A
305	197	C5	F2	C	A	<ESC>!B
306	198	C6	F3	C	A	<ESC>!C
307	199	C7	F4	C	A	<ESC>!D
310	200	C8	F5	C	A	<ESC>!E
311	201	C9	F6	C	A	<ESC>!F
312	202	CA	F7	C	A	<ESC>!G
313	203	CB	F8	C	A	<ESC>!H
314	204	CC	F9	C	A	<ESC>!I
315	205	CD	F10	C	A	<ESC>!J
316	206	CE	Blank1	C	A	None (ignored)
317	207	CF	Blank2	C	A	None (ignored)
320	208	D0	4	P	A	52 (4) or <ESC>Ot
			Tball/Mouse	O	-	None
321	209	D1	5	P	A	53 (5) or <ESC>Ou
			Tball/Mouse	O	-	None
322	210	D2	6	P	A	54 (6) or <ESC>Ov
			Tball/Mouse	O	-	None
323	211	D3	,	P	A	44 (,) or <ESC>Ol (ell)
324	212	D4	Blank3	C	A	None (ignored)
325	213	D5	Blank4	C	A	None (ignored)
326	214	D6	Blank5	C	A	None (ignored)
327	215	D7				
330	216	D8				
331	217	D9				
332	218	DA				
333	219	DB				
334	220	DC				
335	221	DD				
336	222	DE				
337	223	DF				
340	224	E0	BREAK	M,N	U	Short break

Table 4.2 continued

Code Oct	Code Dec	Code Hex	Key	Type	Mode	Code (decimal) or escape sequence sent to host
341	225	E1	7	P	A	55 (7) or <ESC>Ow
			Tball/Mouse	O	-	None
342	226	E2	8	P	A	56 (8) or <ESC>Ox
343	227	E3	9	P	A	57 (9) or <ESC>Oy
344	228	E4	-	P	A	45 (-) or <ESC>Om
345	229	E5	3	M,N	C,CS	None (ignored)
346	230	E6				
347	231	E7				
350	232	E8				
351	233	E9				
352	234	EA	BREAK	M,N	S	Long break
353	235	EB	BREAK	M,N	C	Answerback (see 9.3.2)
354	236	EC	BREAK	M,N	CS	None (enter SETUP)
			SETUP	C	A	" " "
355	237	ED				
356	238	EE				
357	239	EF				
360	240	F0	VDU PIX	M,N	A	None (screen control)
361	241	F1	UP	M	A	<ESC>[A or <ESC>OA
362	242	F2	DOWN	M	A	<ESC>[B or <ESC>OB
363	243	F3	LEFT	M	A	<ESC>[D or <ESC>OD
364	244	F4	RIGHT	M	A	<ESC>[C or <ESC>OC
365	245	F5	MIX	M,N	A	None (screen control)
366	246	F6	PF1	P,N	A	<ESC>[P or <ESC>OP
			CANC OP	P,C	A	<ESC>[P or <ESC>OP
367	247	F7	PF2	P,N	A	<ESC>[Q or <ESC>OQ
			HELP	P,C	A	<ESC>[Q or <ESC>OQ
370	248	F8	PF3	P,N	A	<ESC>[R or <ESC>OR
			MENU	P,C	A	<ESC>[R or <ESC>OR
371	249	F9	PF4	P,N	A	<ESC>[S or <ESC>OS
			Blank6	P,C	A	<ESC>[S or <ESC>OS
372	250	FA				
373	251	FB				
374	252	FC				
375	253	FD				
376	254	FE	SET UP	M,N	A	None (enter Terminal Setup)
377	255	FF				

Codes sent from SBD to keyboard

Certain 8-bit codes are sent from the SBD's peripheral port to the keyboard, to carry out certain VT100 functions such as lighting LEDs, switching keyclick on or off etc. These actions are carried out when the user changes an option in Terminal Setup (see 9.5) or when a particular escape sequence is sent from the host to the SBD. The codes defined below cannot be sent directly from the host.

The least significant bit (bit 0) is used as a selector between LED functions (if set) and other functions (if clear). Bit 7 is the most significant bit (MSB).

Bit	LED Command	Other Command
0	0	1
1	0=online, 1=local LED lit	1=short beep, 0=no effect
2	1=K/B lock LED lit, 0=off	1=long beep, 0=no effect
3	1=L1 LED lit, 0=off	1=keyclick off, 0=keyclick on
4	1=L2 LED lit, 0=off	1=K/B returns code 252(8)
5	1=L3 LED lit, 0=off	ignored
6	1=L4 LED lit, 0=off	ignored
7	1=hidden LED lit, 0=off	ignored

Table 4.3 Codes sent from SBD to keyboard

Pin	Signal (RS232)
1	Tx (to SBD)
2	0v
3	Rx (from SBD)
4	N/C
5	+12v
6	N/C

Table 4.4 Wiring Scheme: Keyboard (Telephone W) to SBD

Note: 1. Pin 1 is the pin furthest from the locking clip.
2. S1 must be linked to 0v at the SBD end of the cable.

4.2 Peripheral Control Unit (PCU)

The Peripheral Control Unit (PCU) is a switching device which is used to connect logically any one of four possible physically connected peripheral types to the Single Board Display.

The physical connections are shown in Figure 2.6, section 2.2.3, and a schematic diagram of the complete system is shown in Figure 1.1, section 1.4. (The keyboard/PCU connection is internal to the keyboard housing).

The required peripheral is selected when the SBD sends a CONNECT command. The PCU is transparent to the user in that the system responds as though the peripheral were connected directly to the SBD.

The PCU is housed inside the SBD keyboard. Communication with the outside world is via 4 serial ports and 1 parallel port. The parallel port is for interfacing to the Tracker Ball whilst the serial lines are for the SBD and the other peripherals.

The local intelligence is provided by a Z80A CPU running at 2.5 MHz. Two Z80A Darts and a Z80A PIO form the four serial ports and one parallel port respectively. A Z80A Counter Timer Circuit generates various clocks and interrupts required by the Z80A CPU and supporting IC's. The firmware is stored in a 4K EPROM with the stack and scratch pad implemented by 2K bytes of static RAM.

NOTE: SBD Models A and B V2.1 firmware requires PCU V2.1 firmware in order to operate correctly when using a trackerball or mouse. All earlier versions of SBD firmware require PCU V1.0 firmware (otherwise a continuous beep is heard when the trackerball/mouse is active).

The user has no access to this area.

The PCU is supplied with power at +12V from the SBD. The keyboard and PCU together take current varying from 0.4A (rest state) to 0.7A (all LEDs lit). When the SBD is switched on with the PCU connected, a reset pulse is generated within the PCU. The PCU then initialises the ports and verifies the correct operation of the static RAM. If the diagnostic check fails, then the PCU hangs and will not respond in any way. This will cause the SBD to timeout when requesting configuration details and the "K/B" string in the power up message will appear in reverse video, after a one-second delay. (See Table 2.3, section 2.2.6.1, and also 2.2.6.3 [B7]).

4.2.1 SBD - PCU protocol

The SBD sends commands to the PCU (see Table 4.5) to tell it which peripheral should be connected. It does this by setting the CON line high, sending the command and then setting the CON line low again after a delay which ensures that the PCU will have received the command. (The purpose of the CON line is to ensure that the 8 bits sent by the SBD to the printer or keyboard cannot be interpreted as control commands. It is also used to determine what is plugged into the keyboard port of the SBD.) A peripheral is "disconnected" by issuing a Connect command for a different peripheral.

CON	Command(octal)	Action by PCU
0	any	No Command
1	000	Send Configuration and Current Routing
1	001	Connect Keyboard & Tracker Ball / Mouse
1	002	Connect Tablet / Table
1	003	Connect Printer
1	004-377	Invalid Commands - Ignored

Table 4.5 Command Codes (SBD to PCU)

When command 0 is received by the PCU, it checks to find out which peripherals are connected. It then sends back a character of 377 (octal) followed by a code to transmit this information and also to say which device was connected last (Table 4.6).

Bit no.	Meaning	
0) 1)	Current Routing (binary)	) 00 - Keyboard & Trackerball / Mouse) 01 - Keyboard only) 10 - Printer) 11 - Tablet / Table
2	Keyboard	0 = Absent 1 = Present (always 1)
3	Trackerball / Mouse	" "
4	Tablet / Table	" "
5	Not Used	
6	Printer	" "
7	Not Used	

Table 4.6 Reply Codes (PCU to SBD)

4.2.2 PCU Connector Wiring

The following tables (4.7 to 4.10) provide all the wiring information necessary to connect peripherals to the PCU. The peripheral must also conform to the relevant communication protocol - see the relevant section in this chapter. For direct connections to the SBD's peripheral port, to SIT, SGT and SBX peripheral ports, and to the MicroPDP/MicroVAX adaptor, see the Specification (1.8). Pin numbering is shown (for the PCU) in Figure 2.6, section 2.2.3. Refer to manufacturers' manuals for pin configurations on the peripherals themselves.

Note: SO, S1 and CON are signals interpreted by the SBD. SA and SB are signals interpreted by the PCU, and are connected to OV on the relevant peripheral. (By this means the PCU detects whether a peripheral is connected). The connectors described are the mating connectors on the cables.

Pin	Signal (RS232)
1	S1
2	+12V in
3	OV
4	Tx (to SBD)
5	Rx (from SBD)
6	N/C
7	N/C
8	N/C
9	CON

Note: SO is connected to CON at SBD end of cable.

Table 4.7 Wiring scheme: PCU (9-way D female) to SBD

Pin	Signal (TTL)
1	OV
2	+5V out
3	Left button (1=down)
4	Right button
5	X+
6	X-
7	Y+
8	Y-
9	Middle button

Table 4.8 Wiring scheme: PCU (9-way D male) to trackerball

Pin	Signal (RS232)
1	OV
2	Rx (from tablet)
3	Tx (to tablet)
4	SB (to OV on tablet)
5	SA (to OV on tablet)
6	N/C

Table 4.9 Wiring scheme: PCU (Telephone A) to tablet

Pin	Signal (RS232)
1	OV
2	Rx (from printer)
3	Tx (to printer)
4	SB (to OV on printer)
5	SA (to OV on printer)
6	S1 (internally conn. to SBD port, pin 1)

Note: Printer must pull S1 low to indicate power-on status

Table 4.10 Wiring scheme: PCU (Telephone W) to printer

4.3 Trackerballs and Mouse

A trackerball, like a digitising tablet, is a device for positioning a cursor on a graphics display. Several types of trackerball are available for use with the SBD: a ready-housed unit (Marconi) with a 2-inch ball (shown in Figure 4.2), and several unhoused units (Penny & Giles) with ball sizes varying from 1.5 inches to 3 inches. A mouse is also available (Penny & Giles), shown in Figure 4.3. A mouse is basically a trackerball held upside down and run along any convenient surface (desk, thigh etc.) to rotate the ball.

The trackerball is rotated in any direction using the fingertips to control some kind of visual feedback, for example a pixel cursor moving in a similar direction. This provides the user with the ability to move the cursor quickly and position it accurately without his eyes leaving the screen. Trackerballs are very well suited to computer aided design applications. One advantage of a trackerball, which is not shared by either a mouse or a tablet, is that the active part of the unit does not move and so can be located by "muscle memory" without having to look or grope for it. This is especially useful where the user frequently has to use both hands for another task.

The three buttons on the Marconi trackerball or the mouse can be used to mark points on the display and can be given specific, user defined functions. The buttons can be pressed in combination to give seven unique codes. Penny & Giles trackerballs do not incorporate any buttons or other sending mechanisms and are intended for incorporation into custom designed consoles rather than stand-alone use. Some buttons on the PCU keyboard are also designated as "send keys" for use in combination with the trackerball or mouse (see CUR in 3.10).

The Marconi ball sits on two steel shafts and a roller bearing. The two shafts drive optical encoder discs which produce trains of pulses for X and Y axes. Incorporated electronics shapes these pulses and indicates forward or reverse rotation. The trackerball produces TTL signals and therefore requires a PCU to interface between it and the SBD. It cannot be connected directly to the SBD.

The Penny & Giles trackerballs and mouse have different mechanisms but a similar electrical interface. They cannot be distinguished by the SBD from the Marconi trackerball, and when connected to the PCU's parallel port, they are all assumed to be trackerballs.

An interface box (Intelligent Serial Interface, or ISI) can be used to connect a trackerball or mouse directly to the SBD. This box converts the parallel interface for the trackerball/mouse into an RS232 serial protocol (see 4.3.3). Any peripheral attached to the

ISI, whether trackerball or mouse, is assumed to be a mouse.

4.3.1 PCU - Trackerball Protocol

The trackerballs and mouse, when the ball is rotated, produce two trains of TTL square wave pulses for each of the X and Y directions. The outputs are quadrature coded so that the direction of rotation is indicated by which train is leading by 90 degrees. (If there is no rotation, flat DC signals are produced).

X1 leading X2 - right
X2 leading X1 - left
Y1 leading Y2 - down
Y2 leading Y1 - up

For the Marconi trackerball, approximately 200 pulses are produced for one full revolution of the ball. The PCU converts these pulses into "pixels moved" at a rate of one pulse per pixel, so that nearly four revolutions are required to move the cursor from one side of the screen to the other at the highest resolution. Faster rotation increases the pulse rate and therefore the apparent speed of the cursor. The Penny & Giles transducers produce similar pulse rates; however, rotating the balls too fast can cause overrun and unexpected cursor motion may result.

The trackerball is such a sensitive device that it is possible to locate a single pixel and yet move the cursor across the screen in only a second or so.

Each button on the trackerball/mouse housing has its own wire in the flying lead (see Table 4.8). This means that combinations of buttons can be distinguished from single-button depressions. Signals are high (0V) when buttons are depressed, low (5V TTL) when not depressed. On the mouse, the terms "left" and "right" assume that the mouse is held with the cable leading away from the user (i.e. the buttons are at the top).

Note that a trackerball or mouse used with a PCU is not interchangeable with one used with an ISI, as the flying lead connectors are differently wired. You must specify the intended use when ordering.

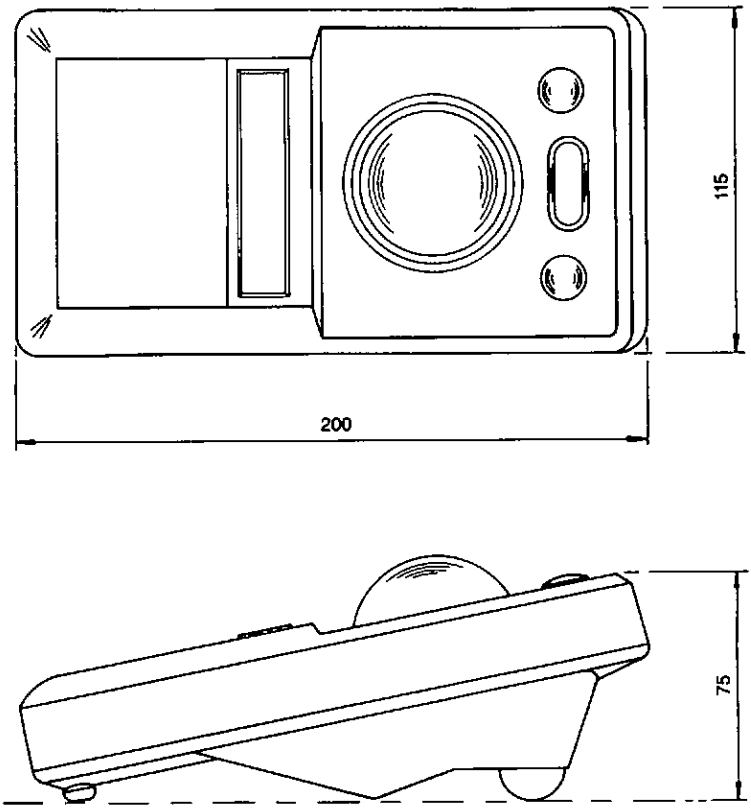


Figure 4.2 Marconi Trackerball (dimensions in mm)

Weight : 600g
Ball diameter: 57 mm (2.25 in)

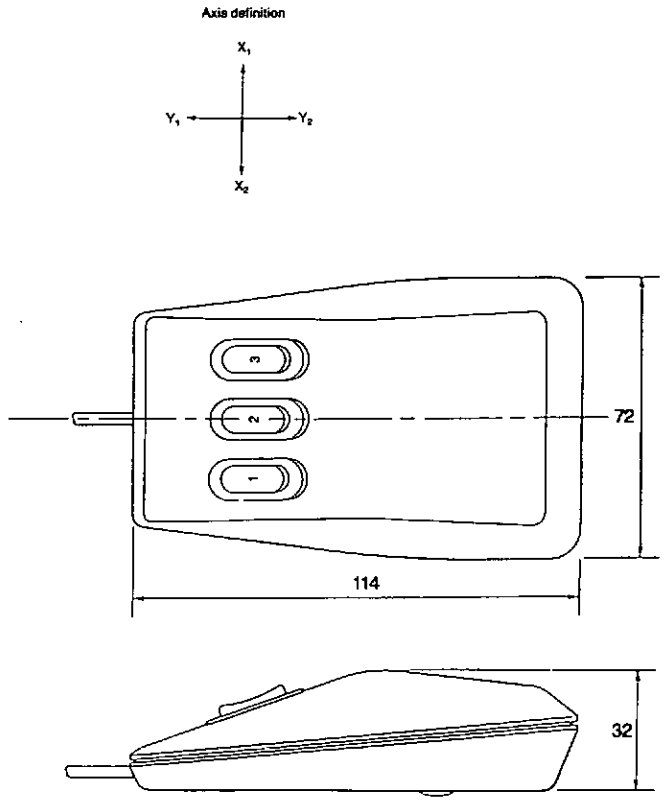


Figure 4.3 Mouse (dimensions in mm)

4.3.2 PCU - SBD Trackerball Codes

The PCU decodes the trackerball output waveforms into the number of pixels moved and the direction of movement. This information is transmitted to the SBD when a count of 7 in either X or Y is reached, or when 80 ms have elapsed and the count is non zero. The data are in the following form (Table 4.11):

Bit no.	Meaning
0)	Number of pixels moved
1)	000 to 111 (0 to 7)
2)	
3	Direction: 0=positive 1=negative
4	Axis : 0=X Axis 1=Y Axis
5)	0) Trackerball
6)	0) Code
7)	1) Identification

Table 4.11 Trackerball codes received by SBD

Note: the positive Y direction for the trackerball is assumed by the SBD to be from top to bottom of the screen, irrespective of the current Y axis orientation selected by SET (see 3.10). Positive X is from left to right.

The above coding leads to 8-bit character values in the range 200 to 237 octal. These do not conflict with codes generated by the keyboard (see Table 4.2).

While the trackerball/mouse is "connected", keycode information is transmitted to the SBD every 80 ms (after the movement code if any), using the following octal codes (Table 4.12):

Buttons			Code	Equivalent
Left	Middle	Right	sent	keypad key
UP	UP	UP	263	0
UP	UP	DOWN	300	1
UP	DOWN	UP	301	2
UP	DOWN	DOWN	302	3
DOWN	UP	UP	320	4
DOWN	UP	DOWN	321	5
DOWN	DOWN	UP	322	6
DOWN	DOWN	DOWN	341	7

Table 4.12 Coding of trackerball / mouse buttons

4.3.3 SBD - ISI protocol

The ISI (Intelligent Serial Interface) is used to interface a trackerball or mouse directly to the SBD's peripheral port (i.e. when no PCU is available). The ISI cannot be connected in combination with another peripheral such as the keyboard.

Consult the manufacturer's documentation for the full ISI protocol. The cable coding tells the SBD that an ISI is connected (see Table 4.1), and on power-up the SBD sends the following bytes (octal) to the ISI to configure it:

```
15      allows ISI to detect baud rate (9600)
<pause>
55      set mode: 8-bit, no ID, displacement, hold at edge, transmit
104     program transmit mode:
3       transmit mode = request, binary, string
102     program Button ID:
0       Button ID (top 5 bits) = 0
```

On connecting the trackerball/mouse, the following bytes are sent to the ISI (octal):

```
46      program Y max:
15      Ymax MSB ) Ymax = 4095 (decimal)
377     Ymax LSB )
45      program X max:
15      Xmax MSB ) Xmax = 4095 (decimal)
377     Ymax LSB )
104     program transmit mode:
0       transmit mode = auto, binary
107     program transmit delay:
24      transmit delay = 78 ms approx.
```

On disconnecting the trackerball/mouse, the following bytes are sent to the ISI (octal):

```
104     program transmit mode:
3       transmit mode = request, binary, string
```

Values returned by the ISI (when in auto transmit mode) occur in groups of 3 bytes at the specified delay rate, as follows:

```
Button (0-7)
X displacement (0-377)
Y displacement (0-377)
```

4.4 Graphics Tablets

Digitising tablets are used to convert absolute positional information into a digital form suitable for use by the SBD. They can be used as general pointing devices to position a cursor, or they can be used to transfer (digitise) diagrams or pictures into the display memory of the SBD. From there, the data can be stored by the host computer for future use or can be output to a printer so that hard copies can be taken.

The tablet itself is a rectangular object with a smooth upper surface, over which is moved a stylus or puck.

Two sizes of digitiser are available for use with the SBD. The desk top version is 40cm x 40cm with an active area of 28cm x 28cm (this is available from two different manufacturers), whilst the other is described as a digitising table and is much larger (active area 91 x 122 cm). Both sizes have a resolution of 0.1mm.

The stylus or puck will operate over paper, plastic film or any drawing surface and is unaffected by pencil lines, ink or conductive plotter output. The normal tablets can operate with the puck or stylus separated by as much as 6mm from the tablet surface. High performance tablets are available which will still work with greater separation to facilitate copying from thick books.

The tablets available are the Summagraphics Bit Pad One (RS232), and the TDS LC-12 (RS232) (illustrated in Figure 4.4). The table is a Summagraphics ID-RS232 with 05-303-1 communications interface board.

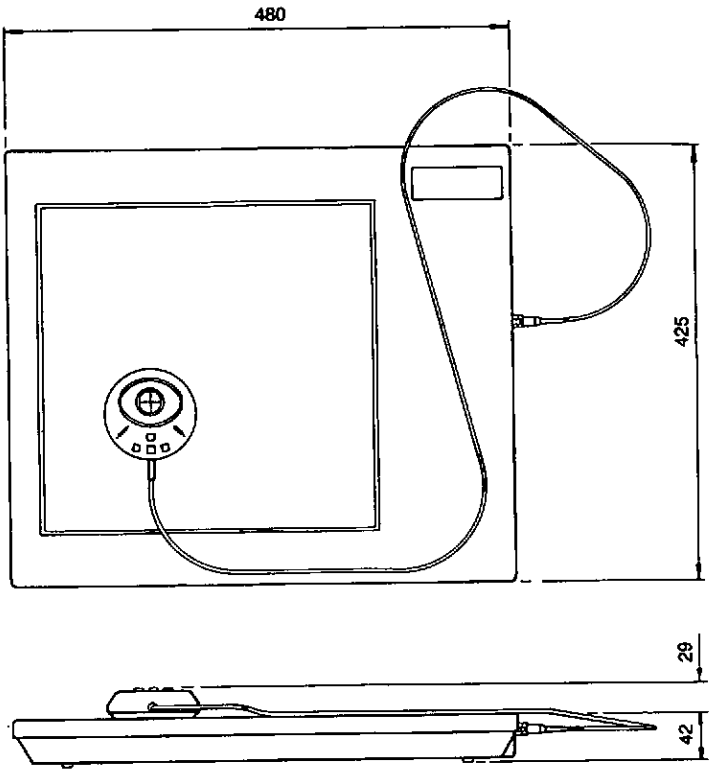


Figure 4.4 TDS LC-12 Tablet

Dimensions in mm.

4.4.1 SBD/PCU - Tablet Protocol

The Summagraphics and TDS tablets transmit the absolute position of the cursor in ASCII using the following format (total 12 characters):

XXXX,YYYY,Z<CR>

XXXX = 4 decimal characters of horizontal coordinates

YYYY = 4 decimal characters of vertical coordinates

Z = 1 hexadecimal character of button data (0 to F). See the TAB command in 3.10 for button/number correspondence.

Both X and Y coordinates are in the range 0 to 2793 (0.1 mm resolution). However, the TDS tablet can be set to a higher resolution if required (0.05 mm - see below) and this gives a range of 0 to 5587 in both X and Y. For both tablets, the origin (0,0) is at the bottom left of the active area.

The tablet is used in a mode in which coordinate pairs are transmitted continuously to the SBD. The SBD uses a buffer to store the data, and sends XON/XOFF to control the flow.

4.4.2 SBD/PCU - Table Protocol

The Summagraphics ID digitising table has a slightly different transmission format from the tablets, though still in ASCII (total 14 characters):

ZSXXXXXSYYYYY<CR>

S = sign (+ or -)

XXXXX= 5 decimal characters of horizontal coordinates

YYYYY= 5 decimal characters of vertical coordinates

Z = 1 hexadecimal character of button data (0 to F). See the TAB command in 3.10 for button/number correspondence.

The range of coordinates for the table is -4572 to +4572 in Y, and -6096 to +6096 in X. The origin (0,0) is in the centre, and X coordinates increase from left to right while Y coordinates increase from top to bottom. (Note this means that the Y axis is inverted with respect to the tablets, whose Y axis increases from bottom to top.)

4.4.3 Setting Up Tablets

Summagraphics Bit Pad One

If the tablet is to be used via a PCU, the factory standard patching is normally adequate. However, if the tablet is to be attached directly to the SBD's peripheral port, it must first be set up for Remote control. (Refer to Section 3.2 of the Bit Pad One manual.) This is done using dual inline (DIL) switches inside the case of the tablet. Unscrew the bottom casing, turn the tablet the correct way up and lift off the surface. Three DIL switches are visible; only Switch 1 (which has 9 switches) and Switch 2 (which has 6) are relevant. Ensure that these switches are set as follows (Table 4.13):

		OFF/-/1	ON/+/0	Meaning
Switch 1 position	1	) Do not alter. If	)	
	2	) changed accident-	)	Factory
	3	) ally, refer to pp.	)	Settings
	4	) 28-29 of manufac-	)	(Calibration)
	5	) turer's manual	)	
	6	Not used		
	7	X		ASCII data format
	8	X		No <LF> (not standard)
	9	X		Metric (0.1mm) resolution
Switch 2 position	1	X	)	
	2	X	)	
	3	X	)	Remote control
	4	X	)	(not standard)
	5	X	)	
	6	X	)	

Table 4.13 Internal switch settings for Summagraphics Bit Pad One

TDS LC-12

As with the Bit Pad One, settings need only be altered if the tablet is to be attached directly to the SBD's peripheral port. (Refer to Sections 2.4.4 and 2.6 of the LC-12 manual.) This is done using dual inline (DIL) switches inside the case of the tablet. Unscrew the bottom casing, turn the tablet the correct way up and lift off the surface. Three DIL switches are visible; only Switch 1 and Switch 2 (which both have 8 switches) are relevant. You can find the locations of these switches in the LC-12 handbook. Ensure that these switches are set as follows (Table 4.14):

		OFF/-/1	ON/+/0	Meaning
Switch 1 position	1		X	)
	2	X		) 9600 baud
	3	X		)
	4		X	) Space
	5	X		) parity
	6		X	8 bit format
	7	X		No <LF> (not standard)
	8		X	Serial
Switch 2 position	1		X	Units mm
	2	0.1 mm	0.05mm	Select as required
	3	Do not alter		
	4	X		) Remote Mode
	5	X		) (not standard)
	6		X	) Sample Rate
	7		X	)
	8		X	)

Table 4.14 Internal switch settings for TDS LC-12

Summagraphics ID

The ID table has a separate controller box containing much of the electronics. This controller has a number of buttons on the front. To operate with the SBD, only the button labelled:

$\overline{\text{SWITCH}}$ STREAM

should be depressed. Note that data is only transmitted to the SBD when the puck is close to the active surface (i.e. when the PROXIMITY light is on).

Normally, internal switches in the controller should not need to be altered. If they are incorrect, they should be set to 9600 baud, BCD format (not normalised), and all format switches off except <CR> format control. Consult the ID manual for details.

4.5 Inkjet Printer

The printer which is available for use with the SBD is the Integrex Colourjet 132GL. This is an inkjet printer, customised to a PPL specification, which has a palette of 36 colours and prints onto a roll of A4 width paper. It can be used in either ASCII text mode, where characters are printed at 30 cps, or in graphics mode where the SBD screen is dumped to the printer bit by bit as binary data. The printer has an actual resolution of 1280 dots per line but an effective resolution of 680 dots per line due to the dithering requirement (see below). At the higher resolutions (768x574) the screen image is put on the paper sideways so that each printer line is a vertical line from the screen.

The mains power connector is at the rear of the unit. The mains power switch is to the right of the unit and there is a control panel on the front (see Figure 4.5). For a full description of the printer and the instructions for use, please refer to the Colourjet 132 handbook.

The SBD has a "transparent" mode which allows the host to interact directly with the printer. This mode is described in 9.4.3.

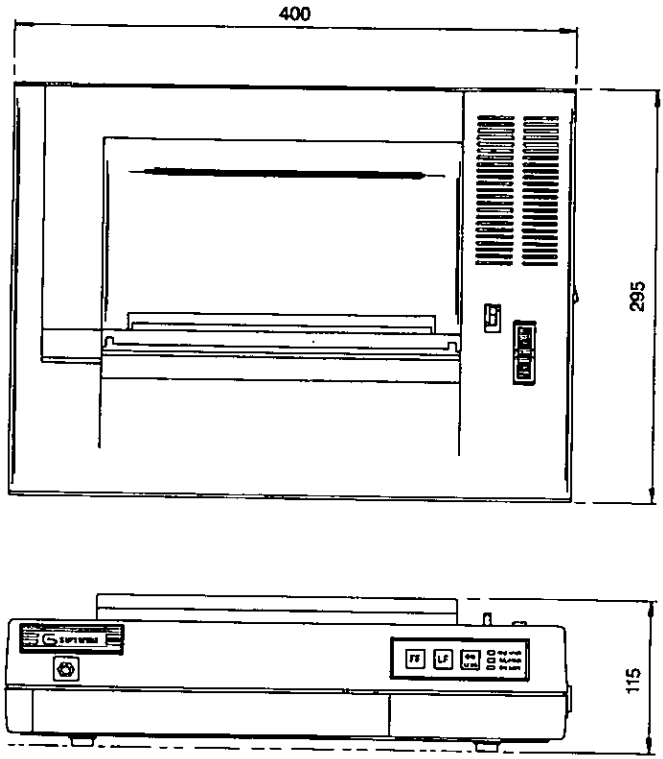


Figure 4.5 Inkjet Printer

Dimensions in mm.

4.5.1 SBD/PCU - Printer Protocol

For dumping graphics data, the printer is used in 1280 dots/line colour graphic Image mode. By combining the (subtractive) primary colours, yellow, cyan and magenta, the printer can generate 8 colours for each dot. Note that in order to produce black, the printer uses black ink rather than combining the three coloured inks. By combining the dots into pairs, a further 28 combinations can be produced. This necessarily means that there is a blank line between each print line to maintain the aspect ratio.

The command <ESC>s is used to select the 16 colours from the 36 available. This command has the format:

<ESC>s P0 P1...P15

where P0 to P15 are 8-bit bytes (no intervening spaces) in the following format:

```
 7 6 5 4 3 2 1 0
+-+-+-+-+-+-+-+-+     B, G and R represent blue, red and green present
¦x B G R¦x B G R¦     respectively (1=present, 0=absent).
+-+-+-+-+-+-+-+-+
```

Each of these bytes represents two adjacent colour dots which will be combined to form the printed colour corresponding to the pixel value (P0 = colour 0, P1 = colour 1 etc.). Since 8 colours are possible for each individual dot, the combined range of colours is 36. See the DMP command in 3.10 for an explanation of the colours produced.

Each colour is then printed by using a 4 bit binary code which is obtained directly from the SBD's 4 bit planes. Two pixels can therefore be defined by a single byte transmitted over the serial line. The command <ESC>t is used to transmit pixel data, as follows:

<ESC>t Nh Nl B0...Bn

Nh and Nl are the high byte and low byte respectively (no intervening spaces) of a binary word defining the number of bytes (n+1) to follow. B0 to Bn are the bytes themselves. Within each data byte, bits 0-3 (LS bits) refer to the right hand pixel and bits 4-7 (MS bits) refer to the left hand pixel. Each such 4-bit nybble serves as an index to the colour mapping table defined by the <ESC>s command. Each pixel is printed using two adjacent ink dots as described above.

See 9.4.3 for the use of the printer in serial mode (SBD transparent mode). By embedding escape sequences in the data stream, you can control the appearance of the printed text. (You can also alter the colour mapping as described above, though it is preferable to use the SDM command (see 3.10) to achieve this.) The printer

powers up in normal black 80-column mode, and is reset to this mode after an SBD pixel dump.

The printer drives input S1 Low to indicate Ready and High to indicate Not Ready. (Not Ready = power off). The PCU passes this signal through directly to the SBD.

Data rate control is provided by the XON/OFF protocol. XOFF (DC3) is sent by the printer when it is switched offline, or its internal buffer becomes nearly full. XON (DC1) is sent when the printer is switched online and on power-up, and when the internal buffer empties.

4.6 Frame Buffer and Hardcopier

The Graftel VP200 Frame Buffer, combined with a Canon inkjet printer, is a device for obtaining hardcopy from a video source. It attaches to the 75 ohm video outputs from the SBD (in parallel with the video monitor) and its operation is therefore independent of the SBD firmware.

The frame buffer has the advantage over the standard Integrex printer that it can capture a video image from the SBD and store it locally in only about 20-30 seconds, after which time the SBD is free to continue while the VP200 prints the captured image on its local printer. Disadvantages of this approach are the extra cost and the loss of the 16-colour mode; also the unit has to be set up manually (using an external ASCII terminal and connecting cable with Rx and Tx signals crossed) so that dynamic remapping of visible to printed colours is not possible. It is the responsibility of the application program to keep the picture static while the image is captured (20-30 seconds).

The frame buffer has a powerful self-setting capability which basically only requires the screen width and height (e.g. 768,574) to be specified. When first setting up, in addition to the external terminal, you will require a simple program which can set the SBD's screen to completely black, then completely white on request. For example:

```
CALL INI(1,0,2,16) ! or appropriate INI command
CALL LIX(0,0)
CALL SOM(0)
CALL MIX(0)
CALL SEL(0,0,0)
CALL RUB           ! set screen black
PAUSE              ! wait for <CR>
CALL SEL(0,0,7)
CALL RUB           ! set screen white
STOP
END
```

See the manufacturer's manual for setting up details.

CHAPTER 5 - CONTENTS

CHAPTER 5 - CONTENTS (continued)

Figures

Tables

5 Command Blocks : Construction and Use

5.1 Introduction

This Chapter describes the structure and use of a "command block", which is a data structure containing a list of graphics commands to be interpreted by the display device. The sources of command blocks are outlined, and software components and interfaces which involve command block handling are described.

This Chapter should be read by anyone intending to create and send their own command blocks (bypassing the host library) or who wishes to write their own DMA device driver for the Single Board Display.

A Command block (CB), sometimes called a "Display List", is a sequential list of binary encoded commands which can be interpreted by a common command executor (in the SBD, part of the monitor firmware, combined with graphics subroutines).

The CB format can easily be generated from any command source (ASCII or binary serial protocols, Ethernet or DMA), so that the command executor is unaware of the source of its CB but is simply passed a start pointer (which may point to a location in the display's local RAM or, using the SBD's DMA interface, in host memory).

The sources of CBs are:

- Serial Line CBs are built by the serial line interrupt service routine as characters are received from the host. The source of the commands is 7-bit or 8-bit characters received on the host serial line. The various serial formats are described in Chapter 7. Commands are converted into the appropriate block command format as specified in 5.4. Very little conversion is required where the host generates the required format, as in the serial Binary protocol.

- DMA CBs can be divided into two kinds as far as the host library is concerned. The host DMA driver and the SBD do not distinguish between the two types.

  Immediate DMA CBs are built by the host library in a data area internal to the library. The command source is the user program calling library subroutines.

  Block DMA CBs are built, or reside, in an array assigned by the user program. They can be built directly by the user (or extracted from storage media) or else constructed by the library in Blocking mode.

- Archived CBs reside in the display's archiving RAM memory. They are built by the display while in archiving mode. The command source is the CB (or CBs) currently being executed, whichever interface it originally derived from.

- Ethernet CBs are copied into a local buffer on board the Ethernet interface piggy-back board, as they are received from the network. The 68000 executes them directly from this buffer. They can be single- or multi-command, as for DMA CB's.

The serial line (ASCII protocol) and immediate DMA CBs are a special case of CB containing only one command per transmission, and once internally converted by the display, only one command per CB. The command executor does not distinguish the number of commands in the block. Block DMA and archived CBs may contain any number of commands, the only constraints being the size of the user array and the amount of available archiving memory respectively, and the maximum size of a command block (32767 words). Serial binary and Ethernet CB sizes are further restricted by the respective protocols used.

5.2 Command Block Structure

5.2.1 General structure

In general a CB contains several "records", each record being one command. The overall structure of a CB is shown in Fig 5.1. The block contains 1 or more commands, each command containing a number of arguments. The maximum number of arguments for any one command at present is 18, but this is not a fixed limit. Some commands may have array arguments, so the number of arguments is not necessarily the number of words in the argument list.

The first word of the CB is reserved for an unsigned byte count giving the size in bytes of the whole CB (including the terminator and the count itself). CBs are thus restricted to 32K words in length. If an error occurs while the CB is being executed, the byte count is overwritten with the offset to the error (see 5.3). NOTE: The bottom bit of this word must always be zero since all valid command blocks are a round number of words long.

All commands begin on a word boundary. This means that commands containing a string (byte array) of odd length are terminated with a null byte. The CB itself must also begin on a word boundary.

One argument (normally the last) in some commands is an array (of bytes or words) rather than a single word. Byte arrays are preceded by a byte count (in a word), and word arrays are preceded by a word count. Byte arrays are held two bytes per word, and are filled to word boundaries with a null byte where necessary (the null is not included in the byte count).

The function code is a byte value in the range 0-255 (decimal). Assigned values are defined in 5.5. The display device cannot assume that the number of arguments is correct or that the command is a valid one, since although the host library performs these checks, it is possible to bypass the library's checks, particularly if the user calls the EGB or OUTPUT subroutines (or performs a DMA I/O via the driver, bypassing the library completely) with a command block which is incorrect. Only the command execution firmware performs argument validation.

Note that the number and meaning of the command arguments in a CB is not necessarily the same as defined in the corresponding host library subroutine call in 3.10. All forms of character string, for instance, are converted to the Hollerith type (i.e. preceded by a byte count); an array may be converted into a number of word arguments (as in SYM, MAT, SDM); and so on. This document alone defines the format of the command arguments as they appear in a CB. However, the meanings of the arguments are only given here where they differ from those in chapter 3.

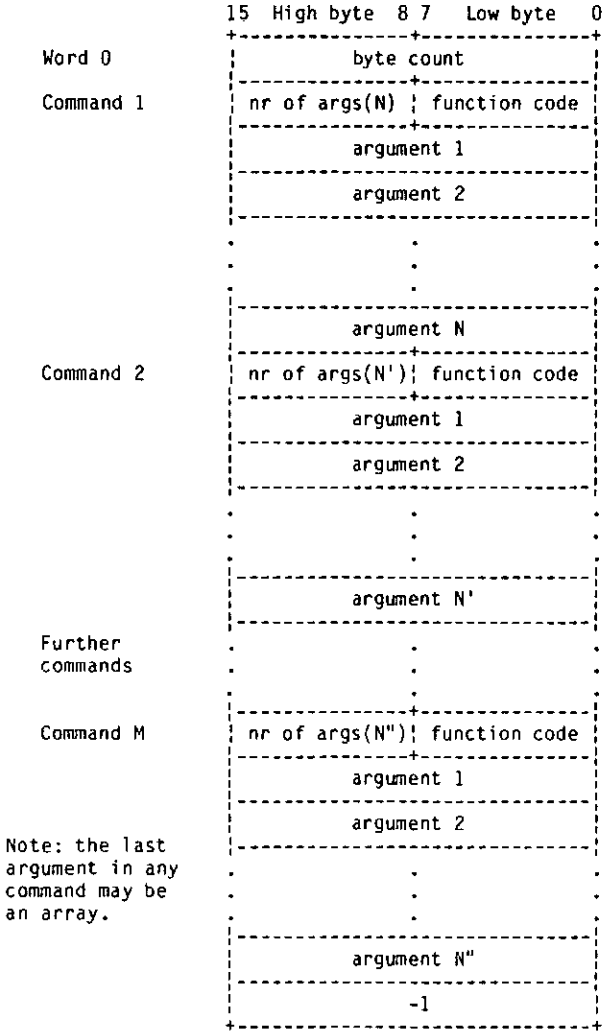


Figure 5.1 General structure of command block

5.2.2 Types of command

There are only two main types of command format:

- Type 1: commands containing integer (word) arguments only
- Type 2: commands containing integer arguments plus an array. This may be a byte array, such as ALP (type 2a), or a word array, such as WIB (type 2b). An array is counted as one argument in the argument count.

Figure 5.2 shows the structure of type 1 commands, Figure 5.3 the structure of type 2a commands, and Figure 5.4 the structure of type 2b commands.

NOTE : In some hosts (and in the SBD's 68000 microprocessor) the bytes are swapped within the word. The PDP-11 addressing scheme is to put the lower byte address in the least significant half of the word. The 68000 does the opposite. For this reason, to allow for any possible host processor, byte arrays are read via DMA as bytes and not as words. The correct format for a block in some non-DIGITAL hosts and in the 68000 has the bytes swapped - each processor does what is natural for itself. The byte reads plus the design of the DMA hardware interface prevent any possible mismatch.

Some commands (e.g. CUR, RIB) require values returned. Space is allowed for these values within the command itself, so that on output to the display the values of these words are not used (though they must be transmitted in the serial formats) but on input they are overwritten. The overwriting is done either by the display itself (DMA interface) or by the host library (serial interface). Note that in some commands an argument can be both input and output.

Some commands which, when called from FORTRAN, take a word array as the last argument (e.g. MAT, SDM, SYM) are converted by the host library into Type 1 commands - all words are copied from the array into the command block and the argument count is increased accordingly. Only in WIB and RIB does the argument count include a word array as one argument.

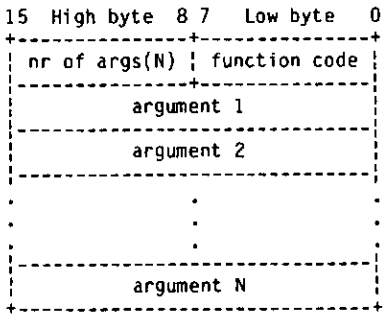


Figure 5.2 Type 1 command structure

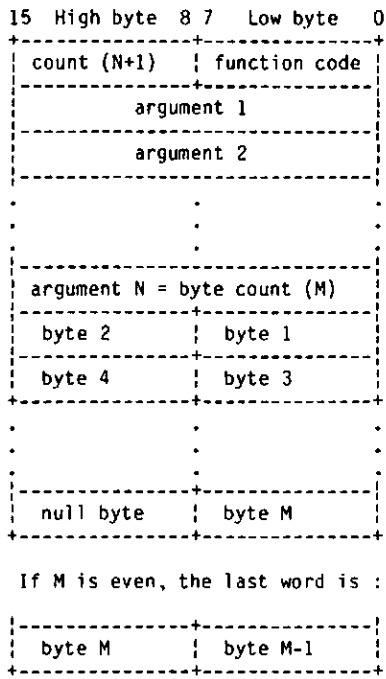


Figure 5.3 Type 2a command structure

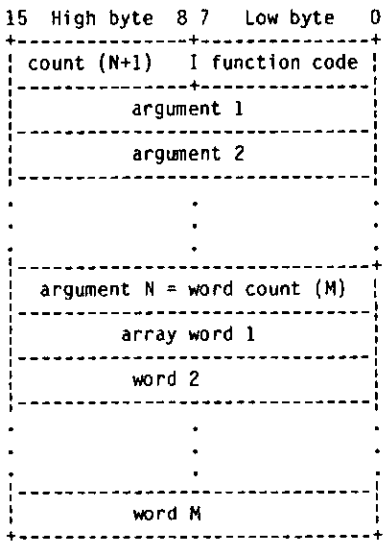


Figure 5.4 Type 2b command structure

5.3 COMMAND BLOCK HANDLING

5.3.1 Host Handling

The host library can generate two kinds of command block to send via the current interface:

- o Immediate mode CBs - contain one command.
- o Block mode CBs - contain any number of commands.

The host library uses the same output routines for each type of block, and the display device is not aware of any difference between the two types.

Immediate mode is the default for the host library if the user has not called BBK. Each command is written into one of two immediate command buffers internal to the library, a -1 terminator is added, and then either the DMA driver is requested to signal the SBD that a block is awaiting execution, or else the block is converted to the appropriate format and sent down the serial line. There are two buffers so that when the library is operating in asynchronous mode, it may be filling one buffer while the display is executing the other.

Any of the commands described in Chapter 3 (except library-local commands such as INI, FIN and the host blocking commands) may appear in an immediate CB generated by the library. The paired commands WIB and RIB are unusual in that the library generates them in immediate CB's, but these are located in user arrays rather than internal library buffers. (They are designed to be interchangeable, i.e. a block created using RIB could be used in a subsequent WIB command to change the area bounds and hence the position of the image block. A block created via either WIB or RIB could be used by the EGB or OUTPUT subroutines.)

Blocking mode is entered when the BBK subroutine is called, and terminated (and the resulting CB optionally executed) by a call to EBK. During blocking mode, each subroutine call loads the appropriate command, word-aligned, into a user-supplied command block (the block address is an argument to BBK). When EBK is called, a terminator (-1) is added after the last command, and the byte count written to the first word. The block may then be saved by the user for later execution by the EGB subroutine, or else executed immediately. The library mode reverts to immediate.

Any command requiring return values (DDI, RIB, TAB, CUR, EAR with one argument, or TRA) is illegal in blocking mode. Certain other library subroutine calls (WIB, BBK, EGB) are also illegal while blocking. EBK is illegal in immediate mode.

The BBK and EBK commands do not appear in the command block. This makes CBs generated by this method indistinguishable from immediate mode CB's.

Users may construct their own CBs in the format already described, without calling library routines, and then execute them using the library calls EGB or OUTPUT (see 3.1). The advantage of using OUTPUT (available only with the assembler coded libraries) is that a program which calls this and no other library subroutine does not include large amounts of code and data tables from the host object library and so is smaller (though this does not apply if the library is configured with an ASCII serial interface or supports full error messages). For such a program the host library effectively acts merely as an interface to send the user's CB to the display device. Users may also write their own library/driver to interface to the SBD as long as the usual command block structure (DMA and serial binary) or command syntax (serial ASCII) is adhered to. (The serial protocols are described in Chapter 7 and the DMA interface in Chapter 6.)

The host library contains checks to ensure that the user cannot block commands into an array which is being accessed asynchronously by the display. Users creating their own blocks should take this possibility into account. See Chapter 3 for a description of the asynchronous mode of the host library.

The basic function of the host library is, therefore, to convert subroutine calls into the standard CB format and invoke different procedures to send them to the display, depending on the interface currently selected. Using the DMA interface, the CB is passed wholesale to the SBD by means of a block pointer. Using the serial binary interface, the library breaks down the CB into sections ("packets") conforming to the size limit of the binary protocol (see Chapter 7.1) but containing as many commands as will fit. The actual command block itself is still sent (nothing is copied to intermediate buffers). However, using the serial ASCII protocol the library breaks down the CB into individual commands, translates them to human-readable ASCII in another buffer and sends that (one command at a time). In both serial protocols the library interface routines are responsible for writing return values into the appropriate sites in the original CB and thence into the Fortran variables.

The above description does not apply to the portable FORTRAN-77 or C serial ASCII libraries. These work by using standard formatted reads and writes and only support immediate mode; they do not generate the standard CB structure.

5.3.2 SBD Handling

DMA interface

When the host wishes a command block to be executed via the SBD's DMA interface, it passes a physical memory address (up to 22 bits) for the block to the SBD via the usual protocol (see Chapter 6). The SBD then accesses the CB via DMA, and the host is free to continue. If the host interrupts the SBD again while a DMA block is being executed, it is ignored, unless the abort bit is set in the appropriate control register.

When the SBD's DMA interrupt service routine is activated and reads the pointer to the CB, it sets a 'busy' flag to inform the monitor idling loop that a DMA CB is waiting for execution. The CB is then executed by calling function-specific subroutines to handle the various commands. Execution of the block is terminated by one of two conditions:

- A -1 terminator is encountered where a function code is looked for (Normal completion).
- No terminator is found at the end of the block (defined by the byte count in the first word of the CB - see Figure 5.1). An error is reported if no previous error has occurred.

If a non-fatal error occurs during CB execution, the error information is stored and execution continues. (Subsequent errors are not stored, but an error count is incremented.)

Fatal errors are those such as "Invalid function code" or "Incorrect number of arguments" which imply that the CB is corrupt, or the SBD has lost synchronisation with it. These errors cause immediate termination of execution.

After termination, the status of the request (zero = success) is placed in bits (0:5) of the Host Command & Status Register (HCSR), from where it is read by the host DMA driver. If an error did occur the command executor will, at the time of the error, have stored some specific status information in the top word of the command block, overwriting the byte count already there (see Figure 5.1). This information is the offset (bytes) from the start of the CB to the start of the command in which the error occurred. If the CB is subsequently re-executed unchanged, therefore, execution will stop before the command which generated the original error, and the error "Invalid command block" will be produced.

The SBD returns other information if errors are found in a CB (see Chapter 6).

The host DMA driver puts the error information into an appropriate status block (in RSX-11M and VAX/VMS, an I/O Status Block (IOSB); in RT-11 a user-supplied status buffer) as described in Chapter 6.

If any CB contains a command requiring return values, the SBD's function subroutine writes these into the command itself. By this mechanism, DMA return values are written directly into host memory without double-handling, but serial line return values are available in local memory for later return down the serial line.

Serial Line Interfaces

The two serial protocols are described in Chapter 7. For each protocol there is a particular character code which signals the SBD to expect transmission of a graphics command or commands.

For the ASCII protocol, the graphics introducer is selectable on power-up (SETUP) and can be changed afterward via the CHI command. It is factory-set as tilde (176 octal).

For the binary protocol the graphics introducer is the 8-bit value 377 (octal). This cannot be changed.

After the ASCII introducer is recognised, the display's serial interrupt service routine interprets further characters 'on the fly' until the command is successfully parsed. As characters are received an internal buffer is filled with the command in normal CB format. Once a carriage return is received signifying the end of the command, if the command has been successfully parsed, the 'busy' flag is set to cause the main firmware loop to execute the internal CB. If any character received cannot be parsed as part of the command, the display reverts to Text mode and subsequent characters are sent to the VDU. The invalid character is not displayed, and no status information is returned to the host. If a line feed occurs immediately after the carriage return at the end of a valid command, it is ignored.

ASCII commands which are syntactically correct but contain semantic errors (Invalid number of arguments, incorrect arguments etc.) are detected by the function subroutines in the SBD firmware, and such errors can be returned to the host. An unrecognised command mnemonic is considered a syntax error.

When the binary introducer is received, the display interprets the following byte as a word count as described in Chapter 7. All subsequent bytes received until the word count is exhausted are copied direct to the internal (256-byte) command block buffer. When the last byte has been copied, a -1 terminator is added, the 'busy' flag is set and the main loop executes the block (which may contain more than one command). Note that there is no attempt to parse the

characters as they are received; any error can only arise when the CB is executed and such errors are returned to the host (if requested). There is no indication of where in the buffer the invalid command was located.

Return values in serial interface CBs are written by the command executor, which maintains two pointers identifying the start and end of the values. (Only one block of return values is allowed, which must be contiguous). When the DMA interface is in use these pointers are not used subsequently, but the serial completion routine uses them to extract the return values, which it converts to the syntax described in Chapter 7.1 and sends to the host, appending a status code if necessary.

Ethernet Interface

If the Ethernet interface option is fitted, graphics command blocks are received fom the network and written to a ring of receive buffers in RAM contained on the piggy-back board. Each command is acknowledged after being processed. When all buffers in the ring are full, packets received are lost until another buffer becomes free. The commands (in the serial binary protocol) are executed by the main firmware loop, and return values (if any) are written back into the receive buffer, from where they are copied to a transmit buffer to be sent back to the originating host.

Return Value Handling

The command executor simply writes return values back into the CB which it is executing, in the appropriate places. If the CB is a DMA one, this is all that needs to be done. If the CB was received via serial line or Ethernet, however, the return values must be extracted from the internal CB and sent back down the same interface. The mechanism used by the SBD is to set up two pointers, one indicating the start of the return values and one pointing to the end. This means that all return values for serial line or Ethernet CBs must be contiguous within the block, and in practice this implies only one return value command in each block.

This is not serious for the serial ASCII protocol which only allows one command per block anyway, but when using the Binary protocol, the host library must terminate and send the packet as soon as any command is encountered requiring return values, forcing such a command to be the last (and only) one in the packet. This method also allows the library to keep track of where return values are found, so that it can write them out to the appropriate program locations if necessary.

Using the Ethernet interface, similar restrictions to the serial binary protocol operate.

Block DMA CB's can contain scattered return value commands, but at the user's risk. The host library prohibits this when in Blocking mode because the library does not know where individual commands are in a block (and so would have no access to the returned values to copy them to the required program locations). However, users creating their own CBs and bypassing the library (or using only the EGB or OUTPUT subroutines) may, if they wish, construct the block with embedded return value commands. They must be aware of the exact construction of the CB in order to extract the values after execution of the block. Neither the library nor the display checks that these commands are alone in a block.

If an attempt is made to archive a command requiring values to be returned, an error is reported and archiving terminated. The command is still executed, however, and all values returned (meaningful or not).

5.4 Command Formats

Each subsection in this section defines the format of a particular command which may occur in a command block.

In all cases, word 0 (the first word of the command) contains the function code in the low-address byte (0) and the number of arguments in the high-address byte (1). The function code is given as a mnemonic (FCxxx); refer to 5.5 for the actual values. All possible values are normally shown for each argument, but this does not imply that all combinations of argument values are valid. See 3.10 for more information.

Optional arguments are given in square brackets. If additional round brackets are shown, this implies "all or none of these arguments must be present".

Valid ranges for X and Y coordinates depend on the currently selected resolution so these are not shown.

The commands INI, FIN, BBK, EBK and EGB affect only the host library and so never appear in a CB.

The type of each command (1, 2a or 2b) is given after the title in brackets. See 5.2.2 for the definitions of command types.

In this section, the term "output values" denotes values which are returned from the SBD to the host.

The byte ordering shown is that for processors such as the PDP-11, which store the least significant byte of a word in the lowest address. Other hosts (e.g. 68000) may have the bytes swapped within a word (which means that byte arrays must have alternate bytes swapped before sending down a serial line to the SBD, which expects to receive the least significant byte first). For the serial binary protocol, the byte ordering must be as defined here throughout, irrespective of the host processor.

5.4.1 ALP - Display text on pixel area (2a)

	Byte 1	Byte 0
Word 0	2 or 4	FCALP
([Word 1	X : start coordinate	])
([Word 2	Y : of text	])
Word 3 (or 1)	Character count (n)	
Word 4 (or 2)	char 2	char 1
	char 4	char 3
	.	.
	.	.
	.	.
EITHER	0	char n
OR	char n	char n-1

5.4.2 BAR - Begin archiving (1)

	Byte 1	Byte 0
Word 0	1	FCBAR
Word 1	Archive file number (0-65534)	

5.4.3 BIT - Draw pixel (1)

	Byte 1	Byte 0
Word 0	2	FCBIT
Word 1	X) coordinates	
Word 2	Y) of pixel	

5.4.4 BLK - Draw solid rectangle (1)

	Byte 1	Byte 0
Word 0	2 or 4	FCBLK
([Word 1	X1 : coordinates of	])
([Word 2	Y1 : opposite	])
Word 3 (or 1)	X2 : corners	
Word 4 (or 2)	Y2 : of rectangle	

5.4.5 BOX - Draw rectangle (1)

	Byte 1	Byte 0
Word 0	2 or 4	FCBOX
([Word 1	X1 : coordinates of	])
([Word 2	Y1 : opposite	])
Word 3 (or 1)	X2 : corners	
Word 4 (or 2)	Y2 : of rectangle	

5.4.6 CHI - Change serial ASCII introducer (2a)

	Byte 1	Byte 0
Word 0	2	FCCHI
Word.1	0	1
Word 2	0	New introducer char

5.4.7 CIF - Draw solid circle (1)

	Byte 1	Byte 0
Word 0	3 or 4	FCCIF
Word 1	Xcent : coordinates of	
Word 2	Ycent : centre of circle	
Word 3	Radius: 1-4095	
[Word 4	Octants: 8-bit pattern]	

5.4.8 CIR - Draw circle (1)

	Byte 1	Byte 0
Word 0	3 or 4	FCCIR
Word 1	Xcent : coordinates of	
Word 2	Ycent : centre of circle	
Word 3	Radius: 1-4095	
[Word 4	Octants: 8-bit pattern]	

5.4.9 CUR - Activate cursor (1)

	Byte 1	Byte 0	
Word 0	2, 3, 4 or 5	FCCUR	
Word 1	X : Initial coordinates	)	These are output values
Word 2	Y : of cursor	)	(words 1 and 2 are also
[Word 3	Transmit key code]	)	input values).
([Word 4	XSIZE) size if])		
([Word 5	YSIZE) box cursor])		

5.4.10 DAR - Delete archive (1)

	Byte 1	Byte 0
Word 0	1	FCDAR
Word 1	Archive file number (0-65534)	

5.4.11 DDI - Get archive information (1)

	Byte 1	Byte 0	
Word 0	9	FCDDI	
Word 1	Directory line - overwritten		
Word 2	) archive file number	)	output values -
Word 3	) number of blocks	)	undefined on input
Word 4	)		
Word 5	)		
Word 6	) as above (total 4 entries)		
Word 7	)		
Word 8	)		
Word 9	)		

5.4.12 DEF - Define channel (1)

	Byte 1	Byte 0
Word 0	4	FCDEF
Word 1	Channel number: 0-15	
Word 2	Mode: 1	
Word 3	Width: 1-4	
Word 4	Group: 0-3	

5.4.13 DLT - Define current line type (1)

	Byte 1	Byte 0
Word 0	1	FCDLT
Word 1	16-bit pixel pattern	

5.4.14 DMP - Dump screen to printer (1)

	Byte 1	Byte 0
Word 0	0, 1, 2 or 6	FCDMP
[Word 1	Timeout : 0-65535]	
[Word 2	Flag : -1, 0 or 1]	
([Word 3	X1 : coordinates])	
([Word 4	Y1 : of opposite])	
([Word 5	X2 : corners of])	
([Word 6	Y2 : rectangle])	

5.4.15 EAR - End archiving (1)

	Byte 1	Byte 0
Word 0	0 or 1	FCEAR
[Word 1	Size of archive (output value)]	

5.4.16 ELF - Draw solid ellipse (1)

	Byte 1	Byte 0
Word 0	4 or 5	FCELF
Word 1	Xcent) coordinates of	
Word 2	Ycent) centre of ellipse	
Word 3	X Radius: 1-4095	
Word 4	Y Radius: 1-4095	
[Word 5	Quadrants: 4-bit pattern]	

5.4.17 ELI - Draw ellipse (1)

	Byte 1	Byte 0
Word 0	4 or 5	FCELI
Word 1	Xcent) coordinates of	
Word 2	Ycent) centre of ellipse	
Word 3	X Radius: 1-4095	
Word 4	Y Radius: 1-4095	
[Word 5	Quadrants: 4-bit pattern]	

5.4.18 FLO - Fill random area with foreground colour (1)

	Byte 1	Byte 0
Word 0	2	FCFLO
Word 1	X) coordinates of seed	
Word 2	Y)	

5.4.19 FSH - Set flashing rate (Model B) (1)

	Byte 1	Byte 0
Word 0	2	FCFSH
Word 1	Time for LUT 0 : 0-65535	
Word 2	Time for LUT 1 : 0-65535	

5.4.20 IDI - Initialise archive directory (1)

	Byte 1	Byte 0
Word 0	0	FCIDI

5.4.21 LIX - Load index registers (1)

	Byte 1	Byte 0
Word 0	2	FCLIX
Word 1	XI - any valid X coordinate	
Word 2	YI - any valid Y coordinate	

5.4.22 MAG - Display text on pixel area (magnified) (2a)

	Byte 1	Byte 0
Word 0	3 or 5	FCMAG
([Word 1	X) start coordinate])	
([Word 2	Y) of text])	
Word 3	Magnification factor: 1-16	
Word 4	Character count (n)	
Word 5	char 2	char 1
	char 4	char 3
	.	.
	.	.
	.	.
EITHER	0	char n
OR	char n	char n-1

5.4.23 MAT - Load mapping tables (Model B) (1)

Format 1 - load pixel mapping table

	Byte 1	Byte 0
Word 0	16	FCMAT
Word 1	LUT select : 0, 1 or 2	
Word 2	value for location 1, 0 - 4095	
.	.	
.	.	
.	.	
Word 16	value for location 15, 0 - 4095	

Format 2 - specify scrolling text colour

Word 0	2	FCMAT
Word 1	2	
Word 2	scrolling text colour 0 - 4095	

5.4.24 MES - Set SBD error handling mode (1)

	Byte 1	Byte 0
Word 0	0 or 1	FCMES
[Word 1	Error mode: 0-63]	

5.4.25 MIX - Display mixed pixel/VDU (1)

	Byte 1	Byte 0
Word 0	0 or 1	FCMIX
[Word 1	Rows of VDU text: 0-24]	

5.4.26 PAG - Select viewing page (1)

	Byte 1	Byte 0
Word 0	1	FCPAG
Word 1	Pixel page number: 1-13	

5.4.27 PIX - Display pixel graphic data (1)

	Byte 1	Byte 0
Word 0	0	FCPIX

5.4.28 RAR - Recall archive (1)

	Byte 1	Byte 0
Word 0	1	FCRAR
Word 1	Archive file number (0-65534)	

5.4.29 RIB - Read image data (2b)

	Byte 1	Byte 0
Word 0	7	FCRIB
Word 1	X1 : upper left corner	:
Word 2	Y1 :	: Bounds of image
Word 3	X2 : lower right corner	: area to be read
Word 4	Y2 :	: from screen
Word 5	Packing mode: 4	
Word 6	Word count N = ((X2-X1+1)*(Y2-Y1+1))/4 rounded up	
Word 7	Location for image word 1	) The next N words are
.	.	) output values.
.	.	) These words
.	.	) are not transmitted by
.	.	) the serial line binary
Word (N+6)	Location for image word N	) format.

5.4.30 RPC - Restore context (1)

	Byte 1	Byte 0
Word 0	0	FCRPC

5.4.31 RUB - Erase rectangle (1)

	Byte 1	Byte 0
Word 0	0 or 4	FCRUB
([Word 1	X1 : Rectangle bounds	])
([Word 2	Y1 : (coordinates of	])
([Word 3	X2 : opposite corners)	])
([Word 4	Y2 :	])

5.4.32 SAP - Select access page (1)

	Byte 1	Byte 0
Word 0	1	FCSAP
Word 1	Pixel page number: 1-13	

5.4.33 SAS - Save context (1)

	Byte 1	Byte 0
Word 0	0	FCSAS

5.4.34 SCA - Set character attributes (1)

	Byte 1	Byte 0
Word 0	4	FCSCA
Word 1	Cell width: 6-20	
Word 2	Cell height: 8-32	
Word 3	Orientation code: 0-3	
Word 4	Character font code: 0-5	

5.4.35 SDM - Set printer colour mapping (1)

	Byte 1	Byte 0
Word 0	16	FCSDM
Word 1	Mapping for colour 0	
.	.	
.	.	
.	.	
Word 16	Mapping for colour 15	

5.4.36 SEL - Select channel (1)

	Byte 1	Byte 0
Word 0	3 or 4	FCSEL
Word 1	Foreground channel number: 0-15	
Word 2	Foreground colour/intensity: 0-15 or -1	
Word 3	Background colour/intensity: 0-15, -1, or 100-115	
[Word 4	Background channel number: 0-15	

5.4.37 SET - Set resolution and pages (1)

	Byte 1	Byte 0
Word 0	1 or 2	FCSET
Word 1	Resolution code: see SET command description	
[Word 2	Number of pages: 1-13]	

5.4.38 SLT - Select predefined line type (1)

	Byte 1	Byte 0
Word 0	1	FCSLT
Word 1	Line type code: 0-7	

5.4.39 SOM - Select output mode (1)

	Byte 1	Byte 0
Word 0	1	FCSOM
Word 1	Video output mode: 0-5	

5.4.40 SSA - Set symbol attributes (1)

	Byte 1	Byte 0
Word 0	1 or 3	FCSSA
([Word 1	Cell width: 1-16	])
([Word 2	Cell height: 1-16	])
Word 3 (or 1)	Orientation code: 0-3	

5.4.41 SYM - Display/define symbol (2a)

Format 1 - display symbol

	Byte 1	Byte 0
Word 0	2 or 4	FCSYM
([Word 1	X : coordinates of	])
([Word 2	Y : symbol	])
Word 3 (or 1)	Char count (n)	
Word 4 (or 2)	char 2	char 1
	char 4	char 3
	.	.
	.	.
	.	.
EITHER	0	char n
OR	char n	char n-1

Note: Characters may only be in the range 'A'-'Z', '*' or space.

Format 2 - define symbol

	Byte 1		Byte 0
Word 0	3 - 18		FCSYM
Word 1	L1 -line 1	) Up to 16 sixteen-bit dot patterns	
[Word 2	L2 -line 2]	) for symbol lines. Not all	
.	.	) bits/words are used depending on	
.	.	) current symbol cell width and	
.	.	) height.	
[Word 16	L16-line 16])		
Word 4-17	0		1
Word 5-18	0		Character
			(in range 'A' - 'Z')

5.4.42 SYN - Synchronise to field (1)

	Byte 1	Byte 0
Word 0	0 or 1	FCTAB
[Word 1	1]	

5.4.43 TAB - Activate digitizing tablet (1)

	Byte 1	Byte 0
Word 0	6 or 7	FCTAB
Word 1	X coordinate transformed	) These are output values.
Word 2	Y coordinate transformed	) Except for Table ID,
Word 3	Table ID	) their values on input
Word 4	Button ID	) are undefined.
Word 5	Wait flag (0 or 1)	) These are input
Word 6	Display cursor flag (0 or 1)	) and output.
[Word 7	Timeout flag (0 or 1) - input only]	

5.4.44 TIN - Initialise digitizing tablet (1)

	Byte 1	Byte 0
Word 0	12	FCTIN
Words 1 & 2	X scale factor	
Words 3 & 4	Y scale factor	
Words 5 & 6	X origin shift	
Words 7 & 8	Y origin shift	
Words 9 & 10	Sine of rotation	
Words 11 & 12	Cosine of rotation	

Note: all arguments to TIN are single-precision floating point numbers (F-format) - see PDP-11 or VAX Architecture Handbook and section 3.5.3.

5.4.45 TRA - Tracker ball sampling ON (1) (Model B only)

	Byte 1	Byte 0
Word 0	3	FCTRA
Word 1	X coordinate	) all values
Word 2	Y coordinate	) are both input
Word 3	Flag value	) and output.

5.4.46 TRO - Tracker ball sampling OFF (1) (Model B only)

	Byte 1	Byte 0
Word 0	0	FCTRO

5.4.47 TXP - Transmit packet (2a)

	Byte 1	Byte 0
Word 0	6	FCTXP
Word 1	Interface selector flag : 0=send via Ethernet non-zero=send via serial line	
Word 2)	) LS	
Word 3)	) Ethernet destination address	
Word 4)	) MS (ignored if Word 1 non-zero)	
Word 5	Message byte count (n)	
Word 6	) Message byte 1	Message byte 0
[Word 7	) .	.]
[.	) .	.]
[.	) .	.]
[Word (n/2+5)	) Message byte n	Message byte n-1]

This command is designed for use in conjunction with the Ethernet Interface Option, though it may still be used on a board with a DMA interface instead. It sends a packet of data via a selectable interface to a specified destination. It can be received via any interface (but if the DMA interface is used, an attempt to forward the data via Ethernet will be ignored). The data cannot be forwarded via DMA since the host must initiate all such transactions. The data sent are words 6 to (n/2)+5. They are not converted in any way. If the data are forwarded via Ethernet, they must include any control information required by the destination, including protocol code, packet type etc. (see Chapter 8). The byte count n must be even, in the range 2 to 248.

5.4.48 TXT - Write text to VDU scrolling region (2a)

		Byte 1	Byte 0
Word 0		2	FCTXT
Word 1		Character count (n)	
Word 2		char 2	char 1
.		.	.
.		.	.
		.	.
	EITHER	0	char n
	OR	char n	char n-1

5.4.49 VDU - Display scrolling text (1)

	Byte 1	Byte 0
Word 0	0 or 1	FCVDU
[Word 1	VDU page operation: -1, 0 or 1]	

5.4.50 VEC - Draw vector (1)

	Byte 1	Byte 0
Word 0	2 or 4	FCVEC
([Word 1	X1) start coordinates	])
([Word 2	Y1) of vector	])
Word 3 (or 1)	X2) end coordinates	
Word 4 (or 2)	Y2) of vector	

5.4.51 WIB - Write image data (2b)

	Byte 1	Byte 0
Word 0	7	FCWIB
Word 1	X1 : upper left corner	:
Word 2	Y1 :	: Bounds of image
Word 3	X2 : lower right corner	: area on screen
Word 4	Y2 :	:
Word 5	Packing mode: 4	
Word 6	Word count N = ((X2-X1+1)*(Y2-Y1+1))/4 rounded up	
Word 7	Image word 1 (4 pixels)	
.	.	
.	.	
.	.	
Word (N+6)	Image word N	

5.4.52 ZOO - Zoom/raster operation (1)

	Byte 1	Byte 0
Word 0	0 - 10	FCZOO
[Word 1	)	]
.	) zoom arguments (see 3.10)	.
.	)	.
.	)	.
[Word 10	)	]

Supplement to Section 5.4 - Private Commands

The following commands are not intended for customer use and should not be present in command blocks. They are PPL test functions and are not subject to the normal customer support.

5.4.53 BGO - Generalised return primitive (BONGO) (1)

	Byte 1	Byte 0
Word 0	5	FCBGO
Word 1	Function selector	
Word 2	)	) These are output arguments.
Word 3	) Return	) The number of meaningful
Word 4	) values	) values depends on the
Word 5	)	) function selector.

This is the only one of the PPL test functions to be implemented by the host library. The FORTRAN interface is:

```
CALL BGO(FLAG,V1,V2,V3,V4)
```

FLAG is a function selector which determines what is returned in V1...V4. (Not all returns may be meaningful; those that are not will be returned unchanged).

The following facilities are currently implemented. PPL reserve the right to add or remove functions without notice.

```
FLAG=0  return resolution / configuration :
        V1 is set to Xwidth e.g. 768
        V2 is set to Yheight e.g. 574
        V3 is set to number of pages (1-13)
        V4 is a bit pattern set as follows:

        bit 0  set=Y origin at lower left of screen
        bit 1  set=second VDU page defined
        bit 2  set=low-resolution SBD, clear=high-res or N.Amer.
        bit 3  set=high DMA address range, clear=low (or Model B)
        bit 4  set=North American SBD, clear=low-res or high-res
        bit 5  set=Ethernet present
        bit 6  set=Model B, clear=Model A
        bit 7
        bit 8  set=keyboard present
        bit 9  set=trackerball/mouse present (via PCU)
        bit 10 set=tablet/table present
        bit 11 set=trackerball/mouse present (via ISI)
        bit 12 set=printer present
        bit 13
        bit 14
        bit 15 set=PCU present
```

BGO (continued)

FLAG=1 return/set current position :
If V1 is non-zero then Xc is set to V3
If V2 is non-zero then Yc is set to V4

Returned values (all modes):
V1 set to zero
V2 set to zero
V3 set to Xc (value before any change)
V4 set to Yc (value before any change)

FLAG=2 read/write SBD location :
V1 MS bits) 32-bit address to be read/written.
V2 LS bits) V2 bit 0 is cleared before using.
If V4 = V1¦V2 (inclusive OR) then write (16-bit) contents of V3 to the address specified by V1,V2. Set V3 to original contents of the address. N.B. cannot write to address zero or to addresses greater than 7FFFFF hex (host memory).
If V4 is not V1¦V2, then read the 16-bit contents of the address specified by V1,V2 into V3.

WARNING: attempts to read/write nonexistent memory locations will cause a bus error trap on the SBD, and the host program will hang.

V4 is cleared, V1 and V2 left unchanged.

FLAG=3 and up : all values returned unchanged. The error "Invalid mode" (11 decimal) is generated.

5.4.54 NOO - Skip words in command block

	Byte 1	Byte 0
Word 0	words to skip (N)	FCNOO

This command will not be found in host library-generated CBs. It causes the command executor to ignore the next N words of the CB, and attempt to interpret the (N+1)th word as the start of the next command. The 'number of words to skip' value does not include word 0. Users generating their own CBs may find this command useful to 'blank out' some primitives, for example by changing FCVEC to FCNOO, a vector can be suppressed without having to rewrite the whole CB.

5.4.55 RST - Simulate power up reset (1)

	Byte 1	Byte 0
Word 0	0	FCRST

This command causes the SBD to execute a simulated power-up reset, including all diagnostics. It is EXTREMELY DANGEROUS to send this via DMA.

5.5 List of Function Codes

This section (Table 5.1) details all the function codes recognised by the SBD or the host library. The SBD recognises only those commands with a "C" in the L/C column.

Key to Table 5.1

Oct/Dec/Hex	Value of code in Octal, Decimal and Hexadecimal bases.
Mnem	Mnemonic for code.
L/C	L = Host library subroutine only. C = Command recognised by SBD firmware L/C = Library subroutine call translated directly into CB command.
Com	Mnemonic or name of command/subroutine.
Meaning	Function of command.

Table 5.1 SBD Function Codes

Oct	Dec	Hex	Mnem	L/C	Com	Meaning
0	0	0	----	not used - reserved to PPL -------		
1	1	1	FCINI	L	INI	Open unit
2	2	2	FCFIN	L	FIN	Close unit
3	3	3	FCSET	L/C	SET	Set resolution and pages
4	4	4	FCDEF	L/C	DEF	Define channel
5	5	5	FCSEL	L/C	SEL	Select channel
6	6	6	FCRUB	L/C	RUB	Erase rectangle
7	7	7	FCLIX	L/C	LIX	Load index registers
10	8	8	FCSAP	L/C	SAP	Select access page
11	9	9	FCPAG	L/C	PAG	Select viewing page
12	10	A	FCSOM	L/C	SOM	Select output mode
13	11	B	FCVDU	L/C	VDU	Display scrolling text area
14	12	C	FCPIX	L/C	PIX	Display pixel graphics area
15	13	D	FCMIX	L/C	MIX	Display mixed pixel/VDU
16	14	E	FCSAS	L/C	SAS	Save context
17	15	F	FCRPC	L/C	RPC	Restore context
20	16	10	FCMES	L/C	MES	Set slave error handling
21	17	11	FCCHI	L/C	CHI	Change serial ASCII command introducer
22	18	12	FCTIN	L/C	TIN	Set digitizing tablet transformations
23	19	13	FCTAB	L/C	TAB	Activate digitizing tablet
24	20	14	FCCUR	L/C	CUR	Activate cursor
25	21	15	FCSLT	L/C	SLT	Select predefined line type
26	22	16	FCDLT	L/C	DLT	Define current line type
27	23	17	FCVEC	L/C	VEC	Draw vector

Table 5.1 continued

Oct	Dec	Hex	Mnem	L/C	Com	Meaning
30	24	18	FCBOX	L/C	BOX	Draw rectangular box
31	25	19	FCBIT	L/C	BIT	Draw pixel
32	26	1A	FCCIR	L/C	CIR	Draw circle
33	27	1B	FCELI	L/C	ELI	Draw ellipse
34	28	1C	FCSCA	L/C	SCA	Set character attributes
35	29	1D	FCALP	L/C	ALP	Display text on pixel area
36	30	1E	FCMAG	L/C	MAG	Display magnified characters
37	31	1F	FCSSA	L/C	SSA	Set symbol attributes
40	32	20	FCSYM	L/C	SYM	Define symbol / display symbols
41	33	21	FCTXT	L/C	TXT	Display text on VDU (archiveable)
42	34	22	FCBLK	L/C	BLK	Draw solid rectangle
43	35	23	FCCIF	L/C	CIF	Draw solid circle
44	36	24	FCELF	L/C	ELF	Draw solid ellipse
45	37	25	FCFLO	L/C	FLO	Random area fill
46	38	26	FCWIB	L/C	WIB	Write image block
47	39	27	FCRIB	L/C	RIB	Read image block
50	40	28	FCBAR	L/C	BAR	Begin archiving
51	41	29	FCEAR	L/C	EAR	End archiving
52	42	2A	FCRAR	L/C	RAR	Recall archive
53	43	2B	FCDAR	L/C	DAR	Delete archive
54	44	2C	FCIDI	L/C	IDI	Initialise archive directory
55	45	2D	FCDDI	L/C	DDI	Display archive directory
56	46	2E	FCEGB	L	EGB	Execute graphics command block
57	47	2F	FCBBK	L	BBK	Begin blocking
60	48	30	FCEBK	L	EBK	End blocking
61	49	31	---- not used - reserved to PPL-------			
62	50	32	FCNOO	C	NOO	Jump words in CB
63	51	33	---- not used - reserved to PPL-------			
64	52	34	FCZOO	L/C	ZOO	Software zoom / raster-op
65	53	35	FCDMP	L/C	DMP	Screen dump to printer
66	54	36	FCSDM	L/C	SDM	Set printer colour mapping
67	55	37	FCMAP	L/C	MAT	Load mapping tables (SBD Model B)
70	56	38	FCFSH	L/C	FSH	Set flashing rate (SBD Model B)
71	57	39	FCRST	C	RST	Simulate power up reset
72	58	3A	FCSYN	L/C	SYN	Wait for field synchronisation
73	59	3B	FCTRA	L/C	TRA	Tracker ball sampling ON (SBD Model B)
74	60	3C	FCTRO	L/C	TRO	Tracker ball sampling OFF (SBD Model B)
75	61	3D	FCBGO	L/C	BGO	Generalised return primitive
76	62	3E	FCTXP	C	TXP	Transmit Packet
77	63	3F	---- not used - reserved to PPL-------			

CHAPTER 6 - CONTENTS

Figures

Tables

6 UNIBUS/Q-Bus Direct Memory Access (DMA)

6.1 Description of DMA Interface

6.1.1 Introduction

The SBD has a Direct Memory Access (DMA) interface to the host computer into which it is plugged. Two DEC bus architectures are supported, UNIBUS and Q-bus (or LSI-11 bus). The SBD can support 16-bit, 18-bit and 22-bit addressing on the Q-bus and 16-bit and 18-bit addressing on the Unibus. 22-bit addressing on the UNIBUS is performed by hardware mapping, and the SBD sees the bus as 18 bits only.

The UNIBUS is found in Digital Equipment Corporation VAX-11, VAX 8600, and PDP series (e.g. PDP-11/34, 11/44, 11/70, 11/84) minicomputers, and the Q-bus is found in MicroVAX, MicroPDP, and LSI-11 series (e.g. PDP-11/23, 11/73) minicomputers. Other vendors sometimes incorporate these architectures into proprietary machines.

The operation of the DMA interface is independent of the host operating system, although for practical purposes some kind of operating-system-dependent software, such as a device driver or handler, is required to allow programs running under the host environment to communicate with the SBD via DMA. Gresham Lion (PPL) supply DMA drivers for the DEC operating systems VAX/VMS, MicroVMS (available for MicroVAX II only), RSX-11M, RSX-11M-PLUS, MicroRSX and RT-11. Drivers can also be supplied for Unix-based systems. These drivers are described in 6.2.

Although in all cases the PPL-supplied drivers may be used directly by user programs to communicate with the SBD (information is provided in 6.2 to allow this), PPL also supply subroutine libraries for all the above operating systems which hide the detail of the driver interaction from the applications programmer. The user interface to these libraries is described in Chapter 3.

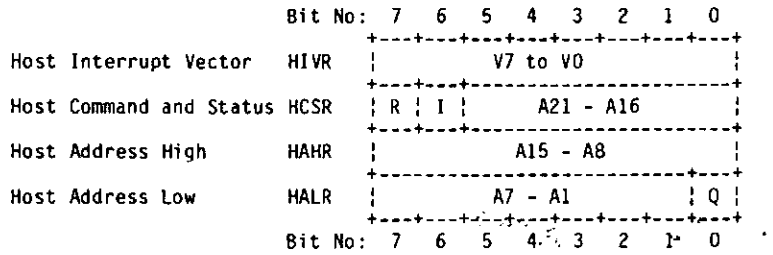
The DMA interface to the SBD is very simple and reliable. Should you be using an operating system not supported by PPL, it is an easy task to write a custom driver for your operating system. Essentially all the driver need do is obtain the physical address of a contiguous or contiguously-mapped buffer containing SBD commands (a "Command Block") in host memory, access the I/O page to communicate this to the SBD, and wait for completion. Drivers can operate under interrupts or by polling. PPL can supply the source code for various implementations of the driver if required; section 6.1.2 provides all the information needed to write a driver except the details of driver interaction with a particular operating system.

The internal structure of Command Blocks is described in Ch. 5.

6.1.2 DMA Interface Programming Information

The SBD's DMA interface is controlled by means of four 8-bit hardware registers which appear at configurable addresses in the I/O page of the host. (See 2.3.2 for details of the range of addresses and how to configure them). The registers appear in consecutive words (least significant byte). The MS 8 bits of each word do not exist. Hence only byte instructions should be used to access them, otherwise the operation of your driver may not be what you expect.

The registers are as follows:



(Q=Abort, R=Ready, I=Interrupt Enable, A=Address, V=Vector)

The normal sequence of operations is as follows:

(1) the host writes HALR and HAHR with command block address bits A0 to A7 and A8 to A15 respectively. Note that A0 = Q (the Abort bit) and must be set to zero at this point. This implies that the command block must be word-aligned.

(2) the host writes HCSR as follows:-

HCSR bits 0:5 : to command block address, bits A16 to A21 (unused MS bits must be zero)

HCSR bit 6 : to zero to inhibit host interrupt on completion
to 1 to enable " " " "

HCSR bit 7 : to zero to initiate the DMA operation.

The act of clearing bit 7 (the Ready bit) of HCSR interrupts the 68000 processor on board the SBD. If the 68000 is currently executing a DMA command block, then the interrupt is ignored. If the 68000 is idle then the new block is queued for execution.

(3) The SBD executes the commands in the command block. When required it requests mastery of the host bus and examines host memory to read command arguments. The SBD firmware adheres to

DEC rules for bus access (maximum 4 word reads in 4 microseconds; at least 4 microseconds between bus requests) so that it does not "hog" the bus. If any command in the block expects return values, the SBD writes these directly into the command block across the DMA interface. There is no restriction on the number of such commands in a block.

(4) When the SBD has executed all the commands in the command block, or encountered a fatal error, it sets HCSR bit 7 to "1" and bits <0:5> to a status code.

It also examines bit 6 (Interrupt Enable) and if this is set to "1", it interrupts the host by writing the value of the interrupt vector in the first register, HIVR. This value is configured by SETUP (see 2.3.2). NOTE: the host driver should never write to HIVR, since this will cause a host interrupt at a vector address corresponding to the value written.

The status code indicates the nature of the first error encounted during the execution of the command block. If execution was error free the status code is zero.

The severity of the status code (see Appendix A) is written to HAHR bits <4:7>. A count of the total number of errors detected in the command block (12 bits) is written to HAHR bits <0:3> (most significant bits of count) and to HALR (least significant bits of count).

If an error or errors were found, the 68000 also writes the first word of the command block with the offset (in bytes) from the start of the list to the first erroneous instruction.

(5) The host may force the 68000 to the idle state at any time during an I/O transaction, by setting bit A0 (the Abort bit, Q) of HALR to "1" and then interrupting the SBD as in (2). It follows that valid command block addresses must be word aligned. All other address bits must be set to zero for an abort operation; the non-zero abort case is reserved for PPL test functions.

When the SBD receives an abort request, it completes the current command (except in the case of commands which may be of long duration such as DMP, TAB, CUR and FLO) and then completes the transaction as in (4) with the status code set to 63 (decimal), i.e. all 6 bits set.

6.2 Host Drivers and Libraries

This section gives details of the DMA drivers available from Gresham Lion (PPL) Ltd. and how user programs may interface directly with them. It also details the extent of PPL-supplied library support for the drivers.

6.2.1 RSX-11M/M-PLUS and Micro-RSX

The DMA driver available for RSX systems (SBDRV) controls devices with the mnemonic SB (e.g. SB0:). It is supplied in source code form in two modules (SBDRV.MAC, SBTAB.MAC). Although the structure of drivers differs between RSX-11M on the one hand and RSX-11M-PLUS and MicroRSX on the other, the driver is able to adapt itself to the host system by means of conditional assembly. The configuration command file GGSBLD.CMD (see 2.4) builds the driver appropriately. The driver can be built to support up to 15 different SB: devices.

Under RSX-11M, the CSR and vector addresses of the DMA devices supported must be specified at configuration time before the driver is built, and cannot be changed thereafter without rebuilding the driver. Under RSX-11M-PLUS and MicroRSX, CSR and vector assignments can be changed dynamically with the CON utility (see 2.4) after the driver has been loaded.

The rest of this subsection gives information on the user interface to the SBD DMA driver for RSX. It is structured in a broadly similar way to the I/O driver descriptions in the RSX-11M I/O Drivers Reference manual. You should read Chapter 1 of that manual if you are not already familiar with the principles of RSX-11M driver interaction.

GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains 0s in all bits for SB: devices. Words 3, 4, and 5 are undefined.

QIO MACRO

Table 6.1 lists the standard and device-specific functions of the QIO macro that are valid for SB: devices.

Table 6.1
Standard and Device-Specific QIO Functions for SB Devices

Format	Function
STANDARD FUNCTIONS	
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
DEVICE-SPECIFIC FUNCTIONS	
QIO$C IO.EGB,...,<stadd,size [,tmo]>	Initiate graphics block execution

stadd

The starting address of a command block (must be word aligned). May be anywhere in the address space of the host.

size

The size of the command block in bytes.

tmo

A timeout value in seconds, in the range 1 to 32767. Defines the time that the driver will wait for I/O completion before aborting the request. If zero, timeout is disabled. If -1 (177777 octal), a configured default value is used (see 2.4).

NOTE: IO.EGB, the device-specific function code, is defined as 400(8). The mnemonic IO.EGB is defined by macro CONFIG in the macro library GGS.MLB which is built at the same time as the SB: driver (see 2.4). To preserve compatibility with possible future releases

of the driver, you should use the supplied mnemonic if possible.

STATUS RETURNS

Table 6.2 lists error and status conditions that are returned by the SB: driver in the first byte (byte 0) of the I/O status block.

If the first byte contains IE.VER, then the second byte (byte 1) contains a code in the range 3 to 77 (octal) giving the first error detected in the command block. (Consult Appendix A for the corresponding errors.) Also, the second word of the I/O status block (bytes 2 and 3) contains a count of the number of errors in the block (bits <0:11>) and a code indicating the severity of the first error (bits <12:14>) as follows:

0 = warning
1 = success
2 = error
3 = informational
4 = fatal

If the first byte of the I/O status block does not contain IE.VER, the second I/O status word is zero.

Table 6.2
RSX SB: Driver IOSB Status Codes

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. For an IO.EGB function, the command block generated no errors.
IE.ABO	Operation aborted. The I/O operation was cancelled by IO.KIL while still in in the I/O queue.
IE.TMO	Operation aborted or timed out. Either the operation was cancelled by IO.KIL while active, or else the operation was cancelled due to the specified timeout expiring.

Table 6.2 continued

Code	Reason
IE.DNA	Device not attached The device specified in an IO.DET function was not attached to the issuing task.
IE.DAA	Device already attached The physical device unit specified in an IO.ATT function was already attached by the issuing task.
IE.DNR	Device not ready Either the specified timeout expired and there was no response to the driver's request to abort the operation, or else the SBD was not ready to process the request.
IE.IFC	Illegal function A function code was specified in an I/O request that is not legal for the SBD DMA driver.
IE.VER	Command block error The SBD executed the command block but the commands in it generated one or more errors. Byte 1 of the I/O Status Block gives the first error encountered.

PROGRAMMING HINTS

The SBD DMA driver does not determine what appears on the screen of the SBD. The key to what is drawn is the collection of commands inside the command block buffer. Consult Chapter 5 for the binary structure of the command block, and Chapter 3 for the meaning and operation of the commands.

Under normal circumstances, the command block is generated by calls to a set of FORTRAN graphics subroutines (see Chapter 3). These subroutines provide a more convenient access to the graphics features of the hardware than do the raw command block instructions.

Aborting a task which uses the DMA interface should cause the SBD to stop what it is doing. If the SBD has failed, there should be a delay of approximately 2 seconds before the task terminates, as the abort request times out. (IO.KIL is automatically generated by an Abort instruction).

6.2.2 RT-11

This section describes the way in which a MACRO applications program can interface directly to the RT-11 DMA handler for the SBD. The DMA handler operates in SJ, FB and XM environments (see 2.4).

The user program interacts with the SBD DMA Handler by means of PROGRAMMED REQUESTS provided by the RT-11 operating system. Programmed requests exist as a user friendly way of accessing facilities within the RT-11 operating system, by means of using software emulator traps (EMT'S). The programmed requests provided by RT-11 are documented in the RT-11 Programmer's Reference Manual.

You should make use of the following programmed requests to interact with the SBD DMA Handler:

.SERR - Prevents fatal error conditions being intercepted by the RT-11 monitor.

.CDFN - Defines the number of RT-11 I/O channels to be used by the host program.

.ENTER - Opens an RT-11 I/O channel through which the user program can communicate with the SBD DMA Handler.

.SPFUN - Initiates the execution of a command block in host memory by the SBD, using the DMA interface.

.WAIT - Waits for I/O activity on the RT-11 I/O channel allocated to the SBD to complete. This request is particularly useful when using the SBD in 'synchronous' mode.

The implementation of these programmed requests within the user program is now described in more detail.

NOTE: The descriptions are given in the sequence in which the user program should call the programmed requests. For additional information, see the RT-11 Programmer's Reference Manual.

.SERR

.SERR is called ONCE, if necessary. It prevents the RT-11 monitor from intercepting 'fatal' error conditions. For example, under normal circumstances, if the .ENTER programmed request failed in the user program, the RT-11 monitor would intercept the error condition, print a ?MON-F-error message on the user's console output device and terminate the program. The GGS host library needs to intercept such errors so that it can print its own error messages. It uses the .SERR programmed request to achieve this.

.CDFN

.CDFN is executed ONCE by the user program if it needs to define additional RT-11 I/O channels. Programs initiated under RT11 are given 16 (decimal) I/O channels by default; the channels are numbered from 0 to 15. The GGS host library must specify additional channels to these, because it is intended for use by FORTRAN programs. The FORTRAN I/O system uses channels 0 to 15 for its own use. The .CDFN programmed request allows the user to specify up to 254 additional I/O channels if required. I/O channel 16 is normally allocated by GGS to interact with the SBD DMA Handler, though this channel number is configurable (see 2.4.2).

Note that in RT-11XM environments, the area of memory containing the addtional channel definition blocks must not be placed within the virtual memory area accessed by Memory Management Unit Register PAR1 (see 3.8.2 for more details).

.ENTER

.ENTER is executed to associate an RT-11 I/O channel (usually channel 16) with the SBD DMA Handler. The .ENTER request 'device block' (dblk) argument contains the name of the SBD unit to be made available for DMA I/O. For example:

- o if DMA communication is required with SBD unit 0 (SB0:), dblk should contain the RADIX-50 characters "SB0";
- o if DMA communication is required with SBD unit 1 (SB1:), dblk should contain the RADIX-50 characters "SB1";

and so on. Up to 7 SB: units can be supported by the driver.

.SPFUN

The .SPFUN (special function) request is provided by RT-11 for users who generate their own customised device handlers. The manner in which the .SPFUN request is utilized is heavily dependent on the device that the request is to support. Therefore, a detailed definition of the implementation of .SPFUN in the context of the SBD DMA Handler is given below.

The .SPFUN Programmed Request for the SBD has the format:

```
.SPFUN  area,chan,func,buf,wcnt,blk[,crtn]
```

where:

area is the address of a six-word EMT argument block.

chan is an RT-11 I/O channel number in the range 0 to 376 (octal), which has already been opened by the .ENTER programmed request.

func is the numerical code of the function to be performed. Mnemonic IO.EGB (200(8)) should be used for this argument value. This mnemonic is defined by macro CONFIG in the macro library GGS.MLB, which is built at the same time as the DMA Handler (see 2.4.2). Gresham Lion (PPL) reserve the right to change this value with future releases of the DMA Handler, so you should always use the mnemonic.

buf is the buffer address (i.e. the start address of the array containing the graphics block to be executed by the SBD). The buf array must be word aligned.

wcnt is the number of WORDS (not bytes) occupied by the command block in the buf array.

blk is the address of a 2-word I/O status block. Upon completion of DMA I/O with the SBD, this block contains information about the status of the I/O operation (see below).

crtn is an optional argument giving the address of the entry point of an I/O completion routine. The crtn facility is used by the GGS host library software.

The format of the blk I/O status block is shown in Figure 6.1. The content is described below.

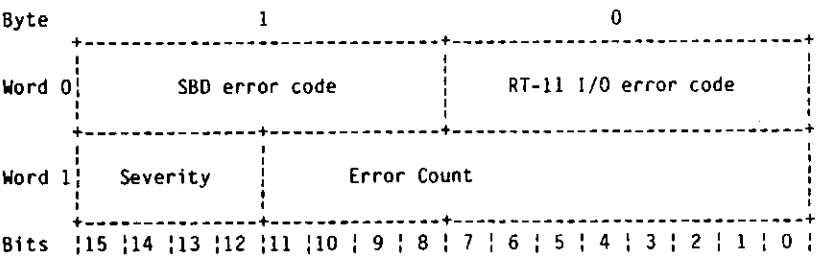


Figure 6.1 RT-11 SB: Handler I/O Status Block

I/O Status Block Word 0

The SBD error code is a "general-code" provided by the SBD firmware. Examples : 0 = SBD operation was error free; 77(octal) = SBD operation was aborted (see 6.1.2). This byte only has significance if byte 0 of the word has the value SEILLC (see below). Codes which can appear in this byte are defined in Appendix A, and are in the range 0-77(8).

The following RT-11 I/O error codes (Table 6.3) are returned in the least significant byte of the first word of the "blk" argument specified in the .SPFUN RT-11 directive :

Mnemonic	No.	Description
SSSUCC	= 1	Success.
SFCHCL	= 200(8)	SBD RT-11 I/O channel is closed.
SFINCD	= 201(8)	Invalid .SPFUN request function code.
SFIUNO	= 202(8)	Invalid SBD unit number.
SFDNR	= 203(8)	Device not ready or not working.
SEILLC	= 374(8)	SBD's error in command block.
SFTIMO	= 266(8)	I/O operation aborted OR timed out.
SEBNWA	= 355(8)	User buffer is not word aligned.

Table 6.3 RT-11 SB: Handler IOSB Status Codes

The mnemonics are defined by macro ECODES in macro library GGS.MLB. Gresham Lion (PPL) reserve the right to change these values, so you should always use the mnemonics where possible.

I/O Status Block Word 1

The Severity Code occupies four bits, and the Error Count twelve bits. Possible Severity Codes are:

0 = warning
1 = success
2 = error
3 = informational
4 = fatal

In the case of errors other than SEILLC, which originate from the handler rather than from the SBD, bytes 1,2 & 3 of the .SPFUN I/O status block are set to ZERO.

Error Count is the number of errors accumulated whilst executing the current graphics block. The error status in Word 0 Byte 1 is that of the FIRST command in the block to generate an error.

.WAIT

The .WAIT programmed request suspends program execution until all I/O requests on the specified I/O channel have been completed. GGS uses the .WAIT programmed request when supporting synchronous DMA I/O with the SBD.

6.2.3 VAX/VMS and MicroVMS

The DMA driver available for VMS systems (SBDRIVER) controls devices with the mnemonic SB (e.g. SBA0:). Since the SBD is a single unit per controller device, the controller letter changes rather than the device number - SBA0:, SBB0:, SBC0: etc. The driver is supplied in source code form (SBDRIVER.MAR) along with a command file to build it. The driver can support as many SB devices as the system configuration allows. The MAXUNITS argument to the DPTAB macro call in the driver code specifies the number of units for the driver to support. By default, it is set to 2. CSR and vector assignments can be changed when the driver is installed (see 2.4).

The rest of this subsection gives information on the user interface to the SBD DMA driver for VMS. Read Chapter 1 of the VMS I/O User's Guide for general information on driver interaction.

Driver Features and Capabilities

The SBD DMA driver provides the following capabilities:

- Execution of display lists (command blocks) in host memory.

Device Information

The user process can obtain SBD device characteristics by using the $GETCHN and $GETDEV system services (refer to VMS I/O User's Guide). The device-dependent data returned is zero.

SBD Function Codes

The only device-dependent function code allowed is IO$_SB_EXEC (32 hex). This maps a buffer in VAX memory containing SBD commands through the SBD's DMA interface. No function code modifiers are permitted. No particular direction of transfer is implied by this function - the buffer may be read, written or both.

The SB_EXEC function code takes the following device/function dependent arguments (P1 through P3) on QIO requests:

- P1 = the starting virtual address of the buffer to be executed
- P2 = the size of the buffer in bytes.
- P3 = a timeout count in seconds. You are advised to set this as large as possible (32767 seconds) for I/O requests which may take some time, such as cursor interaction.

I/O Status Block

The I/O status block format for the execute I/O function is shown in Figure 6.2. Table 6.4 lists the status returns found in the first word of the IOSB. Appendix A lists the message numbers (in the range 0 to 3F hex) and severities of the status returns which may be found in the second word of the IOSB, if the first word contains SS$_NORMAL. Note that these device-dependent status returns are in the normal VMS message format (severity in bits <0:2>, message number in bits <3:14>). The error count gives the total number of errors found in the command buffer. It only occupies the least significant 12 bits of the word.

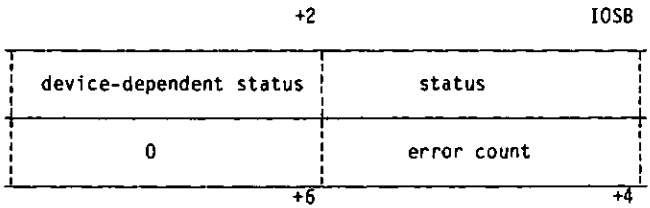


Figure 6.2 VMS SB: Driver I/O Status Block - Execute Function

Table 6.4
VMS SB: Driver IOSB Status Codes

Status	Meaning
SS$_BUFNOTALIGN	Buffer incorrectly aligned. Buffer was not on a word boundary.
SS$_CANCEL	The operation was canceled by the Cancel I/O on Channel ($CANCEL) system service. Applicable only if the driver was actively involved in an execute operation.
SS$_IVADDR	Invalid media address. A buffer address of zero was specified.
SS$_IVBUFLEN	Invalid block. No -1 terminator was found at the end of the block (implied by the byte count in the first word - see Chapter 5).

Table 6.4 continued

Status	Meaning
SS$_NORMAL	Successful completion. The operation specified in the QIO directive was completed successfully. For an execute function, the command block generated no errors.
SS$_POWERFAIL	Power failure occurred during I/O.
SS$_RESET	Device locked or not ready.
SS$_TIMEOUT	Operation timeout. The SBD did not complete the DMA transaction in the specified time.

Programming Example

The following example of a QIO call to the driver is taken from the SBD VMS host library.

```
DMAOUT::
;
; Entry conditions:  R1 contains command buffer address
;                    R2 contains command buffer length
; R0, R1 not preserved.
;
        $QIO_S    ,SBLUN,#IO$_SB_EXEC,SBIOSB,ASTDMA,,(R1),R2,#DMATMO
                                        ; give block to SBD
        BLBS      R0,10$                ; if not OK,
        ERROR     R0,TRAP               ; signal error and crash
10$:    RSB                             ; end of DMAOUT
;
;
;
```

Programming Notes

The GGS library (see Chapter 3) assigns a channel to the SBD automatically (normally when the INI routitne is called). If your process needs to know the channel number, it is at global label SBLUN.

The SBD is a shareable device - more than one process can assign a channel to it without requiring special privilege. However, care must be taken that such processes shut down in an orderly way, by calling FIN if using the GGS library, or by deassigning all channels

to the SBD if not. The reason for this is as follows:

The SBD VMS driver is specially designed so that if a process calls the SYS$CANCEL system service (or uses the $CANCEL macro), then as well as purging any pending I/O for that process, the driver cancels any active DMA to the SBD, even if that I/O was initiated by another process. This is not normal VMS practice, and is designed to allow any process to abort a long-duration primitive (e.g. CUR or DMP) called by another process. The status SS$_CANCEL is returned to the process which initiated the command, e.g.:

```
EXTERNAL SS$_CANCEL
.
.
.
ISTAT = CUR (IX, IY, KEY)
IF (ISTAT.EQ.SS$_CANCEL) GOTO 100
```

6.2.4 UNIX

The UNIX DMA driver will be available at the end of 1985, when this section will be issued.

CHAPTER 7 - CONTENTS

Tables

7 RS232C / RS423 Serial Line Interface to Host

7.1 Binary Transmission Format

7.1.1 Introduction

The Binary serial line transmission format uses 8-bit transmission to provide a compact format and hence short transmission times and more rapid picture update than the ASCII format. In order to operate this protocol, the host computer operating system and hardware must be able to send 8 bit data and pass all 256 possible character codes without interpretation.

7.1.2 Description

Commands in the Binary format are sent in "packets", of length up to 256 bytes, containing a minimum of one command. As many complete commands as will fit can be sent in one packet, but a command cannot be split over two packets. Each command has the normal command block structure described in Chapter 5.

Binary mode is initiated on receipt from the host of the Binary Introducer, code 255(10). The next byte is treated as a word count, where a word is 16 bits made up of two consecutive 8-bit bytes. The value of this byte must be in the range 128(10) to 254(10) and is the one's complement (that is, all 8 bits inverted) of the number of words to follow. The range of the word count is thus from 1 to 127(10).

After the complemented word count, subsequent pairs of characters are received as a block of words until the count is matched, when the SBD reverts to Text mode (characters sent to the VDU text store). In each word the least significant byte is sent first.

It can be seen from a comparison of ASCII and binary protocols (Table 7.1) that data compression is at its best with commands such as VEC involving 2 or more numeric arguments. This is where the greatest efficiency is required. Text commands show no saving on strings.

ASCII coding	Count (bytes)	Binary coding (octal)	Count (bytes)
~VEC 100 200 300 400	21	377 372 027 004 144 000 310 000 054 001 220 001	12
~VEC 100 200	13	377 374 027 002 144 000 310 000	8
~RUB	5	377 376 006 000	4
~ALP 'bye'	11	377 374 035 002 003 000 142 171 145 000	10
~ALP 100 200 'hello'	21	377 371 035 003 144 000 310 000 005 000 150 145 154 154 157 000	16
~CUR 256 512 0	15	377 373 024 003 000 001 000 002 000 000	10
~SYM 48 72 32712 36808 36936 37680 38016 38142 38129 38025 37641 36873 36849 32769 32769 32766 'G'	100	377 354 040 022 060 000 110 000 310 177 310 217 310 220 060 223 200 224 376 224 361 224 211 224 011 223 011 220 361 217 001 200 001 200 376 177 001 000 107 000	40

Table 7.1 Comparison of Binary and ASCII protocols

Notes:

1. The ASCII count includes a carriage return at the end, which is not shown.

2. Although the introducer and word count only occur once at the beginning of a binary packet, there may be more than one binary command in the packet. The coding shown includes the introducer and word count for each command.

7.1.3 Return Data

Once the entire packet has been received, the SBD executes it and returns any output values necessary, plus the block status if requested (see the descriptions of MES and INI in Chapter 3). The status is an error code for the first command error detected, or zero if no errors (see below). Although command arguments destined to contain output arguments are generally meaningless on input from the host, they must be sent anyway as part of the command block. The return value arguments must all be contiguous in the packet, which in practice means that only one command in the packet may have return values.

The syntax for return values and status sent back to the host by the SBD is:

[nnnnn][nnnnn]...[ss](CR)

where :

- each return value (nnnnn) is 5 ASCII digits (with leading zeroes) followed by a space, representing an unsigned decimal number in the range 00000 - 65535;

- the status (ss) is 2 ASCII digits representing a decimal number in the range 00 - 63;

- the string is terminated by carriage return.

(The square brackets enclose optional values.)

If there are no return values and status returns have not been requested, no characters are returned to the host.

The status is zero (00) for success.

This syntax does not apply to the RIB command (see below).

7.1.4 Data Rate Control

Because return data are in 7-bit ASCII, either XON/XOFF or EIA signals can be used for data rate control. However, there is one exception to this, the RIB command (see below).

7.1.5 Design Criteria

The Binary format is designed to allow rapid resynchronisation in the unlikely event of data loss. It should be noted that the protocol is designed for rapid local communication not involving error-prone telephone networks.

Text mode is automatically entered when the word count is exhausted, and it is only in Text mode that the Binary Introducer can be recognised. The Introducer cannot arise in text except as the Introducer. However it can arise in binary mode, particularly as the least significant byte of a coordinate word, where the chance is 1 in 256. There would thus be a small chance of false synchronisation if, due to data corruption, Text mode is entered while the host is transmitting binary.

However, the SBD does not require coordinates greater than 12 bits (unless you are using the 'virtual picture' technique described in 3.7.10) and so the most significant byte will not exceed 15. Rejection of word counts less than 128 prevents this type of false synchronisation. If you are using the SBD's clipping facility via the binary protocol, and data corruption is likely to be a problem, you are advised to set up the SBD's index registers (using LIX) so that that you do not have to send negative coordinates (i.e. restrict yourself to the coordinate range 0 to 32767, so that the most significant byte cannot exceed 127).

7.1.6 RIB - An Exception

Because the RIB command requires the return of large amounts of data, it is treated in a different way from all other commands using the Binary protocol. Most importantly, the return values are in 8-bit binary, not in ASCII, and are not terminated by carriage return. (Status, if required, is two ASCII characters after the return values). RIB is also an exception to the rule that all words in the host command block are sent to the slave; none of the words which will receive the pixel data are sent. There is a restriction on RIB via the serial line in that a maximum of 480 pixels may be read with one command, so a maximum of 240 bytes of return data (plus 1 word for status) can be returned to the host. This restriction is a function of the host software, not the SBD firmware. The SBD will return any amount of data down the serial line, up to the full screen area, if the host serial interface can cope with it.

In 8-bit mode using RIB, the values XON (17) and XOFF (19) can easily occur in the data returned to the host. This means that the XON/XOFF data rate control method is unreliable in these circumstances and you should take care that you:

- do not cause the SBD to generate an XOFF which may be interpreted by your host program as return data for RIB. The SBD sends XOFF when its internal receive buffer contains 256 bytes of unprocessed data, so you should limit the amount of data preceding a RIB so as to avoid this, especially at high data transmission rates. You can send an entire packet of up to 248 bytes, plus another containing the RIB command, without causing an XOFF to be sent, assuming the SBD's buffer was empty to start with.

- do not allow the host operating system to trap these values when using RIB - most DEC operating systems, for example, have a "read pass-all" option on the terminal driver which disables XON/XOFF recognition in 8-bit mode.

- alternatively, disable the XON/XOFF data rate control and use EIA handshaking signals instead.

7.2 ASCII (Human-Readable) Transmission Format

7.2.1 - Introduction

The ASCII serial line transmission format uses 7-bit transmission in the ASCII character set to provide a human-readable syntax (and more importantly, one which can be generated manually).

Under normal circumstances, the binary protocol would be preferred for graphics generation, since it is faster. However, there may be reasons why the binary protocol cannot be used, such as:

- o The host computer serial interface does not support 8-bit data transmission.
- o The host computer operating system traps characters with the 8th bit set, or strips it off, or always sets it, or writes it as a parity bit.

The ASCII syntax is also useful because it can easily be generated from high-level languages such as FORTRAN (see the GGSF77 FORTRAN-77 host library). It can also be typed at a terminal, or edited into a file, and sent to the SBD using the usual operating system utilities (e.g. PIP in RSX-11M).

7.2.2 Description

The syntax of the ASCII format is very simple, and is described here in terms of its constituent elements.

The basic construction of a single ASCII command is:

```
im[sv[,sv...]t
```

where:

- i is the ASCII format Introducer Character. It is configurable for any particular SBD by means of SETUP (see 2.3). The normal factory setting is a tilde (~). It is changed by the CHI command (see 3.10).
- m is the Command Mnemonic, which consists of 3 upper-case alphabetic characters which correspond to the Command Descriptions in Chapter 3 (e.g. VEC, SET etc.). Note that some commands described in Chapter 3 are peculiar to the host library and are not understood by the SBD, such as INI, BBK etc.

s is a separator, which serves to separate command arguments and the first argument from the mnemonic. It may be:

- o one or more space characters (ASCII value 40 octal).
- o one or more horizontal tab characters (ASCII value 11 octal).
- o combinations of the above.

v is a command argument value. This may consist of:

- o an unsigned decimal number in the range 0 to 65535
- o a signed decimal number in the range -32768 to 32767
- o an octal number preceded by a double quote (ASCII value 42 octal) in the range "0 to "177777.
- o an ASCII string, delineated by a leading quote (ASCII value 47 octal). The string may be terminated by another quote or by the command terminator (see below). Non-printing characters, except for space, cause the ASCII command parser to abandon the command. If a single quote is desired in the string, it should be doubled, e.g. 'What''s new'. Note that Hollerith strings (preceded by a character count) are not allowed. This is in contrast to the binary command structure (see Chapter 5) in which all strings are Hollerith.

t is the command terminator, a carriage return (ASCII value 15 octal), optionally followed by a linefeed (ASCII value 12 octal) which is ignored. There may be a separator (see above) between the last argument value and the terminator.

Note that the ASCII syntax for some commands is more easily derived from the command structure in Chapter 5 than from the command description in Chapter 3. For instance, for commands taking a word array argument such as MAT, all elements of the array are separate arguments in the ASCII syntax. For examples of the ASCII syntax see Table 7.1, section 7.1.2.

7.2.3 Return Data

The format of status and return values from the SBD to the host is exactly as for the binary protocol (see 7.1.3). The WIB and RIB commands, however, are not supported via the ASCII protocol, and attempts to use them will lead to either of the errors "Invalid function code" or "Number of arguments error".

7.2.4 Data Rate Control

Either XON/XOFF or EIA data rate control can be used.

7.3 Serial Host Drivers and Libraries

On all the operating systems supported, the GGS library (and any serial line user programs which do not use GGS) employ the standard terminal driver to communicate with the SBD via the serial line. See the INI command (Chapter 3.6) for details of any restrictions imposed by this approach.

7.3.1 RSX-11M/M-PLUS and MicroRSX

Under RSX-11M, in order for the GGS library to use the serial line protocols supported by the SBD, your system must have the full duplex terminal driver with the following SYSGEN options :

(i) Read with no echo
(ii) Control/O cancellation
(iii) Read after prompt
(iv) Variable length input buffering
(v) Transparent read/write

Option (v) above is required by the compressed binary protocol - the ASCII protocol requires the first four options only. It should be noted that although the SBD can receive data from the host and return data via the serial interface at up to 19.2K baud, some serial interfaces (without EIA modem control) cannot reliably receive data at such high speeds. As an example, the DEC DLVJ1 quad asynchronous interface sometimes cannot handle return values transmitted by the SBD at 9600 baud.

When built, the GGS library (module MAINLB) checks for the presence of these SYSGEN options in the system, and produces assembly errors if they are not found.

To use the binary protocol, you must set certain terminal characteristics for the port in use. See 2.4.1.

7.3.2 RT-11

Different RT-11 configurations require the GGS library to be built in different ways. The library configures itself to the system it is built in, by means of the SYSGEN.CND conditional definition file which is created by an RT-11 SYSGEN. On non-SYSGENed systems, where this file is not available, certain defaults are assumed.

Depending on the particular configuration, you may not be able to use certain of the facilities of the library. For instance, unless you have SYSGENed a multiterminal system with Asynchronous Terminal Status Word support and Foreground/Background job capability, you will not be able to use the binary protocol.

7.3.3 VAX/VMS and MicroVMS

There are no special requirements for the VMS serial line software. The standard terminal driver is used by the host library. However, the serial line used must have been made shareable by the system manager, for example:

```
$ SET PROTECTION=WORLD:(RWLP)/DEVICE TXA2:
```

See the SET PROTECTION command in the VMS Operator's Guide.

To use the binary protocol, you must first set certain terminal characteristics for the port to be used (see 2.4.3).

7.3.4 UNIX

The UNIX host software will be released at the end of 1985 when this section will be issued.

CHAPTER 8 - CONTENTS Page

8 Ethernet Interface Option

The purpose of the Ethernet interface is to provide a high speed interface between the SBD and any host for which there is no DMA interface. It is offered as option for the Serial-only version of the SBD. There may or may not be other nodes on the Ethernet network. The Ethernet interface is an alternative to the DMA interface and both cannot be present at once.

The Ethernet option is a piggy-back board which is attached by four screws to the Model B serial-only SBD. (The option is not available for Model A). The piggy-back board connects to the 68000 socket on the SBD mother-board. The displaced processor then mounts on the piggy-back board. The board also contains a 16K byte RAM buffer, Ethernet Controller (LANCE), Manchester Encoder and a Transceiver connector.

8.1 Protocol Description

8.1.1 Introduction

Many of the standard Ethernet protocols are complicated because they support multiple, interlinked, dynamically re-configurable networks. Intelligent Ethernet controllers with firmware to implement the protocols are consequently very expensive. The SBD therefore implements its own private and very simple protocol, to reduce cost, and optimise performance. It is to be expected that all standard protocols will provide access to the physical layer so that users can operate the SBD protocol simultaneously with another protocol on the network.

The SBD Ethernet protocol, because it has been kept as simple as possible, does not guarantee integrity of communication in a multi-user network, neither does it support network routing. Data rate control is provided by an acknowledgement protocol.

8.1.2 Basic Concepts

An SBD may be connected to or disconnected from a particular host. When it is disconnected, it will respond to commands from any host. When it is connected, it will respond only to commands from that particular host. The purpose of a connection is to allow a host to obtain exclusive use of an SBD when there is a risk of multiple hosts competing for its use.

An Abort command allows any host to force the closure of a Connection in an emergency such as a lock-up caused by premature termination of the host process. The current Graphics Packet being executed is abandoned and all queued packets lost. If no Connection has been established, the Abort packet is ignored (though acknowledged).

A host may send Graphics Packets to an SBD via a Connection. Each packet can hold up to 256 bytes of graphical commands (in the GGS serial binary format - see 7.1). If longer packets are sent, the excess data are ignored.

The SBD can buffer 32 packets. Each packet is acknowledged individually when the SBD has finished processing it. (If an Abort is received in the meantime, only the Abort packet is acknowledged). Each Acknowledgement packet contains, among other things, the number of free receive buffers left. The host can combine this information with its own count of unacknowledged packets. The following algorithm is suggested for data rate control:

Let L = number of free buffers
T = total buffers (32)
U = number of unacknowledged packets from this host

Host should stop transmitting when L <= (T-L)/U
Host should resume transmitting when L >= 2(T-L)/U

The quantity (T-L)/U is an estimate of the number of hosts sending packets to the SBD, simplifying to 1 if the host is Connected to the SBD. The assumption is made that all hosts are loading the SBD equally.

Any graphics return data is contained in the Acknowledgement Packet. As described in 7.1, only one command requiring return values may be present in a Graphics Packet, and the maximum amount of return data that may be sent is 120 words (for RIB). In order to write return values to the appropriate program locations, the host may need to suspend further transmission until the Acknowledgement Packet has been received.

When a Connection is in operation, Graphics Packets contain a sequence number after the graphics data. Errors in sequence can thus be detected and reported but no mechanism is available to correct them. When no Connection is in operation, the sequencing mechanism is disabled and the sequence number is not looked for.

The SBD implements the Ethernet Configuration Testing Protocol via its Physical and Broadcast Addresses.

8.1.3 Connections

The Connection facility exists to provide a simple means of checking network integrity and command sequence. A Connection is identified by the host (source) physical address.

A Connection is established by a Connect packet, which initialises the sequence number so that host and SBD are synchronised. If the SBD is already connected, it returns an error

to the Connect packet. When the SBD is connected to one host, all packets from other hosts (except Abort) are acknowledged with an "Already connected" error (see Appendix A for error codes). When the SBD detects an out of sequence packet the default is to report the error, re-synchronise and not execute the packet. These three actions may be suppressed in any combination by a Sequence Error Action code in the Connect packet. The MES graphics command (see 3.10) does not affect the SBD's treatment of sequencing errors, only of graphics commands.

A connected SBD expects the sequence number of the next Graphics Packet received to be one greater than the initial value. From then on, the host sends the sequence number, incremented by one, with each Graphics Packet (after the graphics data). If the Graphics Packet cannot be interpreted, an "Invalid Command Block" error is returned. If the sequence number is not 1 greater than the previous one, the specified error actions occur.

Connections are normally terminated by the host that originated them sending a Disconnect packet. The SBD completes and acknowledges any Graphics Packets in its receive queue before disconnecting (the Disconnect packet is then acknowledged).

Connections can be terminated in an emergency with an Abort packet. This causes the SBD to abandon any Graphics Packet which it is executing, acknowledge it with a "Cancel" status code (77 octal), flush the receive queue, and disconnect. The Abort packet is then acknowledged. The host should time out if the Abort is not acknowledged in a short time (say 5 seconds).

8.1.4 Packet Types and Packet Buffering

The protocol recognises three packet classes: Control Packets, Graphics Packets, and Acknowledgement Packets. Control Packets and Graphics Packets are received by the SBD, which transmits Acknowledgement Packets in response.

Control Packets can be of three types: Connect, Disconnect and Abort. Their action is described in the previous section. Valid Graphics Packets are queued by the SBD; any other packet is examined immediately to see if it is an Abort, or whether it can be ignored (i.e. it is not from the host maintaining the Connection), in which case the action is performed immediately and the receive buffer freed for re-use.

8.1.5 Graphics Packets

Graphics packets (if a Connection is established) contain a sequence count which is maintained by the host and advances by one after each packet is transmitted. The sequence number is initialised by the Connect packet which establishes the Connection. It is assumed that graphics packets will be received and therefore

processed in the order of their sequence numbers. No mechanism is provided in the SBD to correct errors of sequence, due to non-sequential arrival, omissions or duplications. Any sequencing error will be reported in the same way as a graphics error and the erroneous packet will not be executed.

After each Graphics Packet has been processed, an Acknowledgement Packet, containing the corresponding sequence number, is put in the transmit queue. The status value in the Acknowledgement Packet depends on:

(1) the most recent MES command received

(2) the Sequence Error Action in the Connect Packet

If no error was detected in the processing of the Graphics Packet, or if status returns are suppressed by MES, the Acknowledgement Packet contains a code (zero) indicating success. If a Graphics error or an Abort was detected, the Acknowledgement Packet contains an indication of the type of error and its position in the Graphics block. In both cases, if a sequencing error was detected, this overrides the status code in the Acknowledgement Packet (if sequence error reporting is enabled). If return values are required, the Acknowledgement Packet contains a status code and the return values themselves.

The host which establishes a connection and subsequently sends Graphics Packets, should check the Acknowledgement Packets for sequence number. Again, no mechanism is proposed for dealing with errors in sequence, except to report them. A suggested data rate control technique is described in 8.1.2.

The host may have to wait for an Acknowledgement Packet if a Graphics Packet requires return values, to ensure that valid return data is available to the host program before it is allowed to move on.

It is possible for either a Graphics Packet or an Acknowledgement Packet to be lost, in which case there is the possibility of the host waiting for ever to receive an Acknowledgement Packet. It is recommended that the host should have a time-out mechanism to abort and re-establish the connection if an Acknowledgement Packet is unduly delayed. The time-out period should be under the control of the host application program because there is a great variation of acceptable delay. For example, DMP can take several minutes and user interaction with the tracker ball or tablet can take an indefinite time.

8.1.6 VDU Emulation and Serial Line

In Ethernet SBD configurations, these features are retained exactly as in the standard SBD. The SBD with an Ethernet interface can still receive and execute graphics commands received from the serial line.

It is possible to receive scrolling text from Ethernet using the TXT graphics primitive.

A facility exists to send data down the serial line or via Ethernet, and have the data forwarded (either by serial line or Ethernet). The command to do this is TXP (see 5.4 for a description of its structure). If received via serial line, the status (if required) is sent back down the serial line as described in 7.2. If received via Ethernet, the packet is acknowledged in the usual way.

8.1.7 Packet Coding

All packets conform to the Ethernet standard for the level zero transmission medium. This is as follows:-

Byte	Value
0 to 11	Source/Destination Addresses
12 to 13	Protocol Type
14 to 15	Packet Type
)	
)	
)	Packet dependent data
)	Minimum size 44 bytes
)	
n to n+3	Cyclic Redundancy Check (CRC)

The source and destination addresses are Ethernet physical or multicast addresses. The protocol type is a number assigned by Xerox to identify the protocol in use. It must occupy this position in the packet so that it can be found by all protocols.

Two Protocol Type numbers are recognised by the SBD. The first is the number ??-?? (hex), identifying the SBD protocol. The second is the number 90-00 (hex). This signifies the Configuration Testing Protocol. The coding of this Protocol is fully described in the Ethernet Specification and will not be detailed here.

The packet type is the first element of the encoding of the SBD protocol. The following SBD packet types are defined:-

1 Connect
2 Disconnect
3 Abort
4 Graphics

5 Acknowledge

The packet dependent data is peculiar to the SBD protocol and is described fully below. The CRC is the packet checksum which is calculated and verified by the Ethernet controller chip. The theoretical maximum value of n+3 is 1517, but for the SBD protocol, it is 277.

The five packet types are detailed below. Packets containing the code for the SBD protocol and a Packet Type code other than those above are ignored.

1. Connect

Byte	Value	
0 to 11		Source/Destination Addresses
12 to 13		Protocol Type
14 to 15	1	Packet Type
16 to 17		Sequence Error Action
18 to 19		Host Sequence Number (initial)
20 to 59		Padding (ignored)
60 to 63		CRC

Note: Sequence Error Action:
bit 0 set = do not report sequence errors
bit 1 set = do not resynchronise
bit 2 set = execute the packet despite error

2. Disconnect

Byte	Value	
0 to 11		Source/Destination Addresses
12 to 13		Protocol Type
14 to 15	2	Packet Type
16 to 17		Sequence Number
18 to 59		Padding (ignored)
60 to 63		CRC

3. Abort

Byte	Value	
0 to 11		Source/Destination Addresses
12 to 13		Protocol Type
14 to 15	3	Packet Type
16 to 59		Padding (ignored)
60 to 63		CRC

4. Graphics

Byte	Value	
0 to 11		Source/Destination Addresses
12 to 13		Protocol Type
14 to 15	4	Packet Type
16 to 57 (min)		)
to 271(max)		)
		) Graphics Block Format
		)
		)
n to n+1		Sequence Number
n+2 to n+5		CRC

5. Acknowledge

Byte	Value	
0 to 11		Source/Destination Addresses
12 to 13		Protocol Type
14 to 15	5	Packet Type
16		Status (0=success)
17		Offset to Error (0=first command)
18		Error Count
19		Free Buffer Count
20 to 21		Sequence Number
22 to 59 (min)		) Graphics Return Values (words not
to 261(max)		) containing values are undefined).
n to n+3		CRC

8.2 Host Drivers and Libraries

No specific host software is provided by Gresham Lion (PPL) Ltd to support the Ethernet interface option. This is because PPL supply host software for DEC computers only, and it is assumed that where the host is DEC, the DMA interface can be used.

CHAPTER 9 - CONTENTS

Figures

Tables

9 VT100 Features

The SBD will emulate, to the maximum extent permitted by its architecture, the popular VT100 terminal manufactured by Digital Equipment Corporation (DEC). The emulation is sufficient to allow the use of the normal DEC screen editors, KED and EDT.

9.1 VDU Keyboard Layout and Use

The Supervisor SBD VT100 visual display unit (VDU) emulation facility uses a keyboard with a key arrangement similar to an ordinary office typewriter, as shown in Figure 4.1, section 4.1. These keys are supplemented by additional keys to generate command sequences for VDU emulation control, VDU cursor movement, and indicators to convey the current SBD keyboard status. (Note: this section does not describe the layout of the industrial quality Cherry keyboard used for the Supervisor Industrial Terminal (SIT)).

9.1.1 Keyboard Operation

The operator presses keys on the keyboard to cause 8-bit codes to be transmitted to the SBD through its Peripheral Serial Interface. Some keys such as CTRL and SHIFT do not transmit any codes, but modify the codes transmitted by other keys. (See Table 4.2, section 4.1.1).

All code-transmitting keys cause the keyboard to generate an optional key click to verify that the user has performed a keystroke successfully. Keyboard codes are sent to the SBD at the instant a keyboard key is pressed. The keyboard does not wait for the operator to release the key.

The keyboard provides an auto-repeat mechanism. Keys held down for a short time (about 1 second) will cause the keyboard to transmit a continuous stream of character codes for the pressed key to the SBD, until the key is released. All keyboard keys have the auto-repeat facility, with the exception of <SET UP>, <ESC>, <NO SCRL>, <TAB>, <RETRN>, and any key pressed with <CTRL>.

The SBD examines the codes generated by the keyboard, and may or may not send a character or characters to the host as a result. The codes sent are given in Table 4.2, section 4.1.1. Some codes are passed through unchanged, because they happen to correspond to the ASCII code for the character engraved on the pressed key; other codes are modified before being sent. Some keys cause more than one character to be sent to the host - for instance, the keys PF1 to PF4 generate escape sequences. When an ASCII code is sent to the host by the SBD, it is generally echoed back to the SBD by the host terminal driver, and then the character appears on the VDU screen at the current VDU cursor position. The SBD operates in full duplex mode only (no local echoing).

9.1.2 On-line and Local Mode

Interpretation of each ASCII code typed on the keyboard is dependent on the keyboard on-line/local status. The current keyboard status is indicated by the keyboard LEDs ON LINE and LOCAL.

The SBD keyboard is placed in on-line or local mode by:

(1) pressing the <SET UP> key to enter Terminal Setup (see 9.5).

(2) pressing the LINE/LOCAL key (4) to switch between the on-line and local states. The status of the keyboard is indicated by the ON LINE and LOCAL keyboard LEDs, the illuminated LED showing the currently selected state.

(3) pressing the <SET UP> key to exit from Terminal Setup again. The on-line/local keyboard status is preserved.

In local mode, all characters typed on the keyboard are transmitted to the SBD only. No characters are transmitted to the host computer.

In on-line mode, the SBD sends all characters typed on the keyboard to the host computer, if appropriate (see above).

9.1.3 Alphabetic Keyboard Keys

All alphabetic keyboard keys generate lowercase character codes, unless either or both of the keyboard SHIFT keys are pressed down, or the CAPS LOCK key is pressed so that its LED is on. SHIFT and CAPS LOCK modify the alphabetic keys to generate uppercase character codes. CAPS LOCK will only lock the twenty six alphabetic character keys into uppercase mode. Refer to Table 4.2 for the codes generated by the alphabetic keyboard keys.

If the CTRL key is held down when a keyboard key is pressed, it may cause modification of the value sent to the host. Refer to Table 4.2.

9.1.4 Nonalphabetic Keyboard Keys

Each of the nonalphabetic keyboard keys can be used to generate two different codes. Nonalphabetic keys are physically annotated with two nonalphabetic characters arranged in a column. Upon depression these keys generate the code for the lower annotated character. When the nonalphabetic key is pressed with the SHIFT key, the code for the top annotated character is generated by the key. Note that CAPS-LOCK does not affect the nonalphabetic keys; it affects only the alphabetic keys. Refer to Table 4.2 for the codes generated by the nonalphabetic keys.

9.1.5 Keyboard Function Keys

There are several keys on the main keyboard that are designated as function keys. These keys do not produce displayable characters, but are used to control the VDU emulation facility. A description of the function keys follows.

Table 9.1 Keyboard Function Keys

Function Key	Description Of VDU Emulation Function
BACKSPACE	Moves the VDU cursor one character position to the left of the active cursor position, unless the left hand edge of the display screen is encountered.
BREAK	This function key is described below.
CAPS LOCK	When placed in the locked position (red LED on), causes all alphabetic keys to generate an uppercase character code. CAPS LOCK has no affect on nonalphabetic keys.
CTRL	When pressed at the same time as an alphabetic key, generates a VDU emulation control code as described below.
DEL	In local mode, this key code is ignored by the SBD. In on-line mode, this key code is normally used by the host terminal driver to delete the last character entered on the keyboard, and in some cases causes the character to be erased from the display screen, by echoing <BACKSPACE><SPACE><BACKSPACE>
ENTER	A key situated in the keypad section of the keyboard. The functionality of this key is fully described in Section 9.1.3.
ESC	The initial delimiter of an escape command sequence; forces interpretation of the following character string as a command rather than displaying it.
LINE FEED	Moves the VDU cursor from the current cursor position to the same position on the VDU row below. If the cursor is situated at the bottom of the screen when line feed is pressed, the VDU text display scrolls one character position up the screen.

Table 9.1 continued.

Function Key	Description Of VDU Emulation Function
MIX	Enables VDU text and graphics data to be simultaneously displayed on the display screen. Not a standard VT100 key.
NO SCRL	Generates a single XOFF code to inhibit further output to the display screen. When pressed again, generates a single XON code to enable further output to the display screen.
RETURN	Action depends on VT100 "newline" mode (see 9.2). May generate two character codes, carriage return and line feed. Carriage return moves the VDU cursor to the first character position on the current VDU row. Line feed is described above.
SET UP	Places the VDU into Terminal Setup, as described in Section 9.5.
SHIFT	When pressed with an alphabetic key, will generate the uppercase code for the alphabetic. When pressed with a nonalphabetic key, will generate the alternative character code of the two codes assigned to the key.
TAB	Moves the VDU cursor from the active cursor screen position to the next tab position on the current VDU row.
VDU PIX	Allows the entire SBD display screen to show VDU text (VDU mode) or graphics data (PIX mode). Enters VDU if in PIX or MIX mode, otherwise enters PIX mode. Not a standard VT100 key.

The following named keys are applicable to the Cherry type SBD keyboard only, used with the Supervisor Industrial Terminal (SIT).

F1 to F10 Function keys that can be assigned unique significance by the user's application software. The codes generated by these keys are detailed in Table 9.2 below.

Function Key	Associated Codes
F1	ESC!A
F2	ESC!B
F3	ESC!C
F4	ESC!D
F5	ESC!E
F6	ESC!F
F7	ESC!G
F8	ESC!H
F9	ESC!I
F10	ESC!J

Table 9.2 Codes Generated by Cherry Keyboard Function Keys

9.1.6 Break Key

Pressing the BREAK key by itself or in combination with the SHIFT or CTRL keys has the following effect. (Note that there is no BREAK key on the Cherry keyboard - SETUP is entered by pressing the <SETUP> key).

BREAK

Pressing the BREAK key causes the SBD to host computer transmit line to be forced to a zero state (+4 to +6V) for approximately 0.2333 seconds. The DTR (Data Terminal Ready) signal is deasserted during this interval, and reasserted after the interval.

<SHIFT>BREAK

Pressing the BREAK key together with the SHIFT key causes the same action as pressing only the BREAK key, but the break time interval is increased to approximately 3.5 seconds.

<CTRL>BREAK

Pressing the BREAK key together with the CTRL key causes the SBD to transmit a fixed answerback message to the host computer over the SBD to host serial line.

Message Generated (total 16 characters):

SBD nnnnn Xm.nn<CR>

Where :- nnnnn is the SBD serial number (e.g. 01035).
Xm.nn is the firmware type and issue (e.g. A1.10).

<CTRL><SHIFT>BREAK

Pressing the BREAK key together with the CTRL and SHIFT keys causes the SBD to enter SETUP, as described in section 2.3.

9.1.7 Direction/Arrow Keys

Four keyboard keys labelled with arrows are provided on the SBD keyboard. These keys are used in one of three different modes:

a) SBD pixel cursor mode (not standard VT100)
b) VDU cursor mode.
c) VDU application mode.

Arrow Key Function in SBD Pixel Cursor Mode :

The SBD pixel cursor is selected and (optionally) displayed on the SBD display screen by utilising one of the SBD cursor control commands (e.g. CUR or TRA). When the pixel cursor is active, the cursor can be moved around the screen by pressing the appropriate arrow key. When pressed, each arrow key emits a continuous stream of cursor direction codes using an auto-repeat hardware mechanism provided by the keyboard. The SBD processes the pixel cursor movement information generated by the arrow keys to provide dynamic software gearing of pixel cursor movement. This facility can only provide vertical or horizontal movement of the cursor and is intended as a fall-back if no more convenient method of cursor control (e.g. a mouse or trackerball) is available.

Arrow Key Function in VDU Cursor Mode :

VDU cursor mode is enabled by sending the escape sequence ESC[?1l (one-ell) to the SBD VT100 emulator. In VDU cursor mode, each arrow key generates the following characters sent to the host:

Cursor Arrow Key	Code Produced By Key
Right arrow	<ESC>[C
Left arrow	<ESC>[D
Up arrow	<ESC>[A
Down arrow	<ESC>[B

Note that these escape sequences are identical to the corresponding (default) cursor movement escape sequences (CUF, CUB, CUU, CUD) described in 9.2.2. This explains why the Arrow Keys move the cursor when in Local mode.

Arrow Key Function in VDU Application Mode :

VDU application mode is enabled by sending the escape sequence ESC[?1h to the SBD VT100 emulator. When application mode is enabled, the arrow keys on the SBD keyboard produce the following escape sequences:

Keyboard Arrow Key	Code Produced By Key
Right arrow	<ESC>OC (oh)
Left arrow	<ESC>OD
Up arrow	<ESC>OA
Down arrow	<ESC>OB

Application mode allows the user to assign unique significance to each arrow key within the user's application environment.

9.1.8 Auxiliary Keypad

A block of 18 keys on the right hand side of the SBD keyboard (see Figure 4.1) is designated as an auxiliary keypad. The keypad consists of keys for the numerals 0 to 9, .(period), ,(comma), -(minus sign), four semiprogrammable function keys marked PF1 to PF4, and an ENTER key.

The auxiliary keypad is used in one of three modes:

a) Numeric mode.
b) Application mode.
c) Pixel cursor mode.

Auxiliary Keypad Numeric Mode

Keypad numeric mode is enabled by sending <ESC>> (escape followed by a right chevron) to the SBD VT100 emulator. In numeric mode, keys located on the auxiliary keypad section of the SBD keyboard normally generate the ASCII codes for the numerals 0 to 9, decimal point, minus sign, and comma. In addition, the ENTER key generates the same code as for the RETURN key. Thus it is impossible to distinguish between pressing a key on the auxiliary keypad, and pressing the corresponding key on the main keyboard.

In Keypad Numeric Mode, the keys PF1..PF4 generate the escape sequences <ESC>[P, <ESC>[Q, <ESC>[R, and <ESC>[S respectively.

Auxiliary Keypad Application Mode

If the origin of key codes common to both the keypad and the main keyboard is significant, the auxiliary keypad can be set into application mode by sending <ESC>= to the SBD VT100 emulator.

Application mode sets each keypad key to generate an escape code sequence as listed in Table 9.3 below:

Key	Keypad Application Mode	Key	Keypad Application Mode
0	<ESC>Op	-(minus)	<ESC>Om
1	<ESC>Oq	,(comma)	<ESC>Ol
2	<ESC>Or	.(period)	<ESC>On
3	<ESC>Os	ENTER	<ESC>OM
4	<ESC>Ot	PF1	<ESC>OP
5	<ESC>Ou	PF2	<ESC>OQ
6	<ESC>Ov	PF3	<ESC>OR
7	<ESC>Ow	PF4	<ESC>OS
8	<ESC>Ox		
9	<ESC>Oy		

Table 9.3 Auxiliary Keypad Codes Generated For Application Mode

Note: sequences contain letter "O" not numeral zero. Comma sequence contains lowercase "L" not numeral 1.

Auxiliary Keypad Pixel Cursor Mode

The SBD pixel cursor is enabled and displayed using one of the SBD cursor control commands (e.g. CUR or TRA) described in 3.10. When the SBD is performing a pixel cursor operation, the auxiliary keypad keys are programmed to provide a means of formally terminating the cursor session. The code generated from the keypad key pressed to terminate the cursor session is made accessible to the user's application software by using a subroutine argument value.

In pixel cursor mode, the auxiliary keypad key codes are mapped as described in Table 9.4 below.

Key	Cursor Code	Key	Cursor Code
0	0	.(period)	10
1	1	PF1	11
2	2	PF2	12
3	3	PF3	13
4	4	PF4	14
5	5	-(minus)	15
6	6	,(comma)	16
7	7	ENTER	17
8	8		
9	9		

Table 9.4 Auxiliary Keypad Codes Generated For Pixel Cursor Mode

Note: These key codes are not affected by pressing SHIFT, CAPS LOCK, or the CTRL keys in combination with keypad keys.

9.1.9 Keyboard LED Indicators

Seven LED (light emitting diode) indicators are provided on the SBD keyboard to display keyboard status information. The name and significance of each LED is described below (Table 9.5):

Keyboard LED Name	Purpose
ON LINE	When illuminated, this LED indicates that the SBD keyboard is on-line to the host computer. Characters can be sent to the host when keys are pressed.
LOCAL	When illuminated, this LED indicates that the SBD keyboard is off-line or in Local mode. Characters typed on the keyboard are not sent to the host computer.
KBD LOCKED	Indicates that the SBD cannot send characters to the host and its internal keyboard buffer is full. Further key depressions cause the keyboard bell to sound.
L1 to L4	Provided for user application use. These LEDs can be manipulated by using control sequences described in Section 9.2.2.

Table 9.5 Keyboard LED Indicators

9.2 Standard VT100 Features

The following sections describe the standard VT100 compatible control features provided by the SBD VT100 emulator.

9.2.1 Control Characters

Control character codes reside in the numeric range zero to 37(octal)/1F(hex), and 177(octal)/7F(hex). Control characters can be typed on the SBD keyboard by pressing certain keys together with the CTRL key, or else by certain dedicated keys. Control characters can also be sent to the SBD from the host computer. The control characters supported by the VT100 emulator are described in Table 9.6 below.

Notes on Table 9.6:

1. (*) The NO SCROLL key generates XOFF/XON alternately.

2. Column 4 gives the key to be pressed on the SBD keyboard (in conjunction with the CTRL key) to send the character to the host. Where a labelled key is given in column 4, this key operates alone (without the control key.)

3. Some operating system terminal drivers convert these control characters into other characters according to the currently set terminal characteristics. For instance, under RSX-11M, if the terminal is not set /FORMFEED, the terminal driver converts the <FF> character into four line feeds before sending it to the terminal.

Table 9.6 Control codes supported by the SBD VT100 Emulator

Control Char	Octal Code	Hex Code	<CTRL> plus	Action taken on receipt from host
NUL	000	00	@	Ignored on input (not stored in terminal buffer).
ENQ	005	05	E	Transmit SBD answerback message (see sections 9.1.6 and 9.3.2).
BEL	007	07	G	Sound bell tone from keyboard.
BS	010	08	H or BACK SPACE	Move cursor to the left one character position, unless it is at the left margin, in which case no action occurs.
HT	011	09	I or TAB	Move the cursor to the next tab stop or to the right margin if no further tab stops are present on the line.
LF	012	0A	J or LINE FEED	Move the cursor down the screen from the current column position, to the same column position on the next VDU line. If the cursor is at the bottom of the screen, the VDU text is scrolled up.
VT	013	0B	K	Interpreted as LF.
FF	014	0C	L	Interpreted as LF.

Table 9.4 continued

Control Char	Octal Code	Hex Code	<CTRL> plus	Action taken on receipt from host
CR	015	0D	M or RETURN	Move cursor to the left margin on the current line.
SO	016	0E	N	Invoke G1 character set, as designated by ESC) control sequence.
SI	017	0F	O	Select G0 character set, as selected by ESC(control sequence.
XON* (DC1)	021	11	Q	Causes terminal to resume transmission.
XOFF* (DC3)	023	13	S	Causes terminal to stop transmitting all codes except XOFF and XON.
CAN	030	18	X	If sent within an escape control sequence, the sequence is immediately terminated and is not executed.
SUB	032	1A	Z	Interpreted as CAN.
ESC	033	1B	ESC	Introduces an escape control sequence.
DEL	177	7F	DELETE	Ignored on input (not stored in input buffer).

9.2.2 VT100 Control Sequences

This section describes the standard ANSI VT100 control sequences supported by the SBD VT100 emulator.

The control sequences can be transmitted from the host computer to the SBD, or can be entered on the SBD keyboard in LOCAL mode (Refer to Section 9.1.2).

Cursor Position Report - VT100 To Host CPR

<ESC>[Pn;PnR default value: 1

Solicited by <ESC>6 (DSR), CPR reports the active cursor position to the host computer by means of two parameter values. The first parameter specifies the line and the second parameter the column in which the active VDU cursor currently resides. The default condition with no parameters, or parameters of 0, is equivalent to a cursor at home position.

The numbering of VDU lines depends on the state of the Origin Mode selection (DECOM).

Cursor Backward - Host To VT100 And VT100 To Host CUB

<ESC>[PnD default value: 1

CUB moves the active cursor position Pn places to the left, across the screen. If Pn is zero or 1, the cursor is moved one position to the left across the screen. Attempts to move the cursor to the left of the left screen margin results in the cursor staying at the left screen margin. Produced by the Left Arrow key in Cursor Key Mode.

Cursor Down - Host To VT100 And VT100 to Host CUD

<ESC>[PnB default value: 1

CUD moves the active cursor Pn positions down the screen, without altering the column position. If Pn is zero or 1, the active position is moved one line downward. Attempts to move the cursor below the bottom screen margin results in the cursor stopping at the bottom screen margin. Produced by the Down Arrow key in Cursor Key Mode.

Cursor Forward - Host To VT100 And VT100 To Host CUF

<ESC>[PnC default value: 1

CUF moves te active cursor position Pn places to the right, across the screen. If Pn is zero or 1, the active cursor position is moved one character position to the right. Attempts to move the cursor to the right of the right screen margin results in the cursor stopping at the right screen margin, irrespective of whether Autowrap ison or off. Produced by the Right Arrow key in Cursor Key Mode.

Cursor Up - Host To VT100 And VT100 To Host CUU

<ESC>[PnA default value: 1

CUU moves the active cursor Pn positions up the screen without altering the cursor column position. If Pn is set to zero or one, the active cursor is moved one character position up the screen. Attempts to move the cursor above the top screen margin results in the cursor staying at the top screen margin. Produed by the Up Arrow key in Cursor Key Mode.

Cursor Position - Host To VT100 CUP

<ESC>[Pm;PnH default value: 1

CUP moves the active cursor position to VDU line Pm, VDU column Pn. A value of zero or one for either Pm or Pn results in the cursor being moved to the first line or column in the display, respectively. CUP with no parameters is equivalent to a move cursor to home action.

Device Attributes - Host To VT100 DA

<ESC>[Pnc default value: 0

The host computer sends DA to obtain a device attributes control sequence from the SBD VT100 emulator. DA is sent with no Pn parameter, or Pn set to zero.

The SBD VT100 emulator responds to the DA request by sending the following DA control sequence back to the host computer :-

<ESC>[?1;0c "This device is a VT100 with no options present."

Keypad Application Mode - Host To VT100 DECKPAM

<ESC>= default not applicable

The DECKPAM control sequence sets the SBD keyboard keypad section keys to generate control sequences, as detailed in Section 9.1.8.

Keypad Numeric Mode - Host To VT100 DECKPNM

<ESC>> default not applicable

The DECKPNM control sequence sets the SBD keyboard keypad section keys to generate ASCII codes corresponding to the characters engraved on the keys, as described in Section 9.1.8.

Load LEDS - Host To VT100 DECLL

<ESC>[Psq default value: 0

Set programmable LEDs L1 to L4 according to the DECLL control sequence Ps parameter(s).

Parameter	Parameter Meaning
0	Clear LEDs L1 through to L4.
1	Light L1.
2	Light L2.
3	Light L3.
4	Light L4.

LEDs L1 to L4 are annotated on the SBD keyboard.

Restore Cursor - Host To VT100 DECRC

<ESC>8 default not applicable

DECRC causes the previously saved cursor position, graphic rendition, and character set to be restored.

Save Cursor - Host To VT100 DECSC

<ESC>7 default not applicable

DECSC causes the current active cursor position, graphics rendition, and currently selected character set to be saved (See DECRC).

Set Top And Bottom Margin - Host To VT100 DECSTBM

<ESC>[Pm;Pnr default values: see below

This sequence sets the top and bottom margins to define a scrolling region. Pm is the number of the top VDU line in the scrolling region; Pn is the number of the bottom VDU line in the scrolling region.

If no parameters are specified, the default scrolling region is designated as the entire screen, without margins. (Lines 1 to 24).

The minimum size of scrolling region allowed is two VDU lines ; the top margin must be numerically less than the bottom margin.

This control sequence also moves the active cursor to the home position. (Also see description for the Origin Mode control sequence DECOM).

Erase in Display - Host To VT100 ED

<ESC>[PsJ default value: 0

ED erases all or some of the characters in the display according to the value of parameter Ps.

After the erasure, the active cursor position remains intact.

Parameter	Parameter Meaning
0	Erase from the active cursor position to the end of the screen inclusive, (default).
1	Erase from start of the screen to the active cursor position, inclusive.
2	Erase all lines of the display. The active cursor position is not changed.

Erase in Line - Host To VT100 EL

<ESC>[PsK default value: 0

Selectively erases characters in the current active VDU line, according to the Ps parameter value.

After the erasure, the active cursor position remains intact.

Parameter	Parameter Meaning
0	Erase from active cursor position to the end of the line, inclusive (default).
1	Erase from the start of the screen to the active cursor position, inclusive.
2	Erase all of the current VDU line inclusive.

Horizontal Tabulation Set - Host To VT100 HTS

<ESC>H default not applicable

Set one horizontal tab stop at the active cursor position.

Horizontal And Vertical Position - Host To VT100 HVP

<ESC>[Pm;Pnf default value: 1

HVP moves the current active cursor position to the VDU line specified by Pm, and the VDU character position specified by Pn. A parameter value of zero or one in either Pm or Pn, or an omitted parameter, results in the active cursor position being placed on the first VDU line or VDU column of the display, respectively.

HVP with no parameters moves the active cursor position to the cursor home position.

The numbering of lines and columns is dependent on the state of the Origin Mode; refer to description of control sequence DECOM.

Index - Host To VT100 IND

<ESC>D default not applicable

IND causes the active cursor position to be moved down the screen one character position, without changing the cursor column position. If the active cursor position is at the bottom of the screen, a scroll up is performed.

Next Line - Host To VT100 NEL

<ESC>E default not applicable

NEL causes the active cursor position to be moved to the first character position on the next VDU line downward. Attempts to move the active cursor position below the bottom screen margin results in a scroll up being performed.

Reverse Index - Host To VT100 RI

<ESC>M default not applicable

RI causes the active cursor position to be moved to the same horizontal position on the preceding VDU line. Attempts to move the active cursor position above the top screen margin result in a scroll down being performed.

Reset Mode - Host To VT100 RM

<ESC>[?Ps;Ps;...;Psl (lower-case L) no default values

RM resets one or more VT100 operational modes, as specified by each selective parameter. See also Set Mode (SM).

Parameter	Parameter Meaning
1	Reset SBD keyboard arrow keys to produce escape control sequences for VDU cursor control. Refer to Section 9.1.7.
7	Turn off VDU text wraparound function ("autowrap"). The reset wraparound state causes any displayable characters received when the cursor is at the right hand margin of the screen, to replace any previous character there.
20	Causes the interpretation of the line feed character to imply only vertical movement of the active cursor position, and causes the RETURN key to send only a single carriage return code when pressed.

Select Character Set - Host To VT100 SCS

The character sets G0 and G1 can be assigned to three different types of character sets. The G0 character set is enabled when the SI (shift in) code is received; the G1 character set is enabled when the SO (shift out) code is received. Default on power-up is G0.

G0 Sets	G1 Sets	Meaning
<ESC>(A	<ESC>)A	Select United Kingdom character set.
<ESC>(B	<ESC>)B	ASCII character set.
<ESC>(0	<ESC>)0	Special graphics character set. Refer to Section 9.2.3 for more information.

The United Kingdom and ASCII character sets conform to the ISO international standard. The Special Graphics character set is a DEC character set; the characters associated with codes 137 (octal)/5F(hex) to 176(octal)/7E(hex) are replaced with the special graphics characters. The current character set is selected until another SCS escape control sequence is received.

Select Graphic Rendition - Host To VT100 SGR

<ESC>[Ps;...;Psm default: 0

SGR enables the graphic rendition specified by the parameter(s). All subsequent characters received by the SBD VT100 emulator are rendered according to the SGR parameter(s) until the next SGR command is received.

Parameter	Parameter Meaning
0	Switch all graphics rendition attributes off.
4	Underscore all subsequent characters sent to the VDU screen.
7	Display all subsequent characters sent to the VDU screen in reverse video form.

Set Mode - Host To VT100 SM

<ESC>[?Ps;...;Psh no default values

SM sets one or more VT100 operational modes, as specified by the Ps parameter(s).

Parameter	Parameter Meaning
1	Set SBD keyboard arrow keys to produce escape control sequences for application mode. Refer to Section 9.1.7.
7	Enable VDU text wraparound function ("autowrap"). The wraparound set state causes all displayable characters received when the active cursor is at the right hand screen margin, to advance the active cursor to the first character position on the next line, scrolling the screen up if required and permitted.
20	Causes interpretation of the line feed character to imply moving the active cursor to the first character position on the next line. The RETURN key is also set to generate two character codes; carriage return and line feed.

Tabulation Clear - Host To VT100 TBC

<ESC>[Psg default value: 0

Parameter	Parameter Meaning
0	Clear the horizontal tab stop at the active cursor position.
3	Clear all horizontal tab stops.

9.2.3 Special Graphics Characters

The SBD VT100 emulator can be set into special graphics mode by using the SCS (Select Character Set) escape control sequence, as described in Section 9.2.2. Special graphics mode causes the characters associated with character codes 137(octal)/5F(hex) through to 176(octal)/7E(hex) to be replaced according to Table 9.7 below.

Octal Code	Hex Code	Graphic With US or UK set	Special Graphics Character
137	5F		Blank
140	60	◆	Diamond
141	61	a	Checkerboard (error chr.)
142	62	b	HT Horizontal tab
143	63	c	FF Form feed
144	64	d	CR Carriage return
145	65	e	LF Line feed
146	66	f	Degree symbol
147	67	g	Plus/minus
150	68	h	NL New line
151	69	i	VT Vertical tab
152	6A	j	Lower-right corner*
153	6B	k	Upper-right corner*
154	6C	l	Upper-left corner*
155	6D	m	Lower-left corner*
156	6E	n	Crossing lines*
157	6F	o	⎺ Horizontal line - scan 1*
160	70	p	⎻ Horizontal line - scan 2*
161	71	q	─ Horizontal line - scan 3*
162	72	r	⎼ Horizontal line - scan 4*
163	73	s	⎽ Horizontal line - scan 5*
164	74	t	Left "T"*
165	75	u	Right "T"*
166	76	v	Bottom "T"*
167	77	w	Top "T"*
170	78	x	Vertical bar*
171	79	y	Less than or equal to
172	7A	z	Greater than or equal to
173	7B	{	Pi
174	7C	¦	Not equal to
175	7D	}	UK pound sign
176	7E	~	Centered dot

Table 9.7 Special Graphics Characters on SBD VT100 Emulator

Note: line-drawing characters marked * do not join up horizontally or vertically - see 9.3.2.

9.3 Unsupported VT100 Features

This section details features that are not supported on the SBD VT100 terminal emulation implementation, and VT100 features that differ from the DEC implementation of the VT100 terminal.

The main reasons for unsupported features are architectural. The SBD is intended mainly as a high-resolution graphics device, not a VDU, and the VDU emulation is provided for convenience only.

To conserve display memory, the VDU lines are not mapped contiguously to the screen, which means that there are horizontal lines between the VDU rows which are always black. For this reason, the special graphics line-drawing characters (see 9.2.3) do not quite join up, there is no light screen background facility, reverse video appears in discrete lines, and double-width or double-height characters are impractical.

Furthermore, the VDU text font is 5x7 (horizontal rows repeated on some resolutions) rather than 7x9 as in the VT100. This affects the horizontal line-drawing characters. The blank pixel column between characters is left blank, so that line-drawing characters do not quite join up horizontally either.

As only one memory plane is allocated to the VDU display, character attributes which would require more, such as half-intensity and flash, are not supported.

In general, the conclusion to be drawn from this is that if you want to do graphics, use the pixel display not the VDU.

9.3.1 Unsupported Control Sequences

Generally, these escape sequences are parsed correctly but ignored. They are not echoed on the VDU display.

Line Size (Double Height and Double Width) Commands

Note: On some computer systems, the hash sign is printed as #. This should be taken into consideration when refering to the following escape sequences.

<ESC>#3 Change active VDU line to double height top half (DECDHL).

<ESC>#4 Change active VDU line to double height bottom half (DECDHL).

<ESC>#5 Change active VDU line to single width, single height (DECSWL).

<ESC>#6 Change active VDU line to double width, single height (DECDWL).

These command sequences are not supported on the SBD due to the blank lines between VDU rows and columns. They are ignored.

Character Attributes

<ESC>[1m Display all subsequent characters in bold type (SGR).

<ESC>[5m Display all subsequent characters as blinking characters (SGR).

These command sequences are not supported on the SBD due to there being only one memory plane for the VDU display. They are ignored, but any other character attributes in the same escape sequence are actioned. (Reverse video and Underline attributes are supported).

Character Sets

<ESC>(I Select alternate character ROM for G0 character designator (SCS).

<ESC>)1 Select alternate character ROM for G1 character set designator (SCS).

<ESC>(2 Select alternate special graphics character ROM for G0 character set designator (SCS).

<ESC>)2 Select alternate special graphics character ROM for G1 character set designator (SCS).

These command sequences are not supported due to ROM storage space limitations.

Support VT52 or ANSI Compatible Terminal

<ESC>[?2h Set VT52 VDU emulation mode. Interpret and execute VT52 compatible escape sequences (SM).

<ESC>[?2l Set ANSI VDU emulation mode. Interpret and execute ANSI compatible escape sequences (RM).

Support for VT52 computer terminal emulation is not provided by the SBD.

80 or 132 Columns of VDU Text

<ESC>[?3h Select 132 columns of VDU characters across the display screen (SM).

<ESC>[?3l Select 80 columns of VDU characters across the display screen (RM).

The internal representation of VDU text in the SBD provides for only 80 columns of VDU text to be shown on the display screen (64 in some resolutions on the low-resolution version).

VDU Text Scrolling Mode

<ESC>[?4h Select smooth scroll mode (SM).

<ESC>[?4l Select jump scroll mode (RM).

The method of accessing VDU text stored in internal format within the SBD does not allow this facility to be provided.

Screen Display Mode

<ESC>[?5h Display the entire VDU screen as black characters on a white background (reverse video) (SM).

<ESC>[?5l Display the entire VDU screen as white characters on a black background (normal) (RM).

The internal representation of VDU data within the SBD and the provision for character descenders would cause a black line to appear between each VDU row when reverse video is selected. Thus this feature is not supported.

Screen Origin

<ESC>[?6h Select relative origin mode (SM).

<ESC>[?6l Select absolute origin mode (RM).

The method of accessing VDU text stored in the internal format within the SBD does not allow this facility to be provided.

Interlaced Mode

<ESC>[?9h Select VDU display screen interlaced mode (SM).

<ESC>[?9l Select VDU display screen non-interlaced mode (RM).

The SBD does not allow the VDU text display raster to be independent of the graphics display raster. This facility is not provided. The raster is either interlaced or non-interlaced depending on the currently selected display resolution.

Graphics Processing Option

<ESC>1 Enable graphics processor option.

<ESC>2 Disable graphics processor option.

The SBD has inherent pixel graphics capability. Therefore this facility is not supported.

Terminal Reset

<ESC>c Reset VT100 terminal (RIS).

The SBD VT100 emulator does not support this facility, as it is dangerous if the DMA interface is active at the time. There is a command in the SBD command set (RST) to achieve the same thing, which is less likely to arise by accident.

Confidence Tests

Note: On some computer systems, hash is printed as #. This should be taken into consideration when referring to the following escape sequences.

<ESC>#8 Fill screen with uppercase E's for screen alignment (DECALN).

<ESC>[2;Psy Perform the following diagnostic tests (DECTST).
Ps = 1 - Power-up self test.
2 - Data loop back test.
4 - EIA modem control test.
8 - Repeat selected test indefinitely.

The SBD VT100 emulator does not support these facilities. The power-up self-test can be initiated by entering Terminal Setup and hitting the RESET key (0) - see 9.5, by entering and leaving SETUP - see 2.3, or by sending the RST command - see 5.4.

VT52 Terminal Compatibility

The SBD does not emulate the DEC VT52 type terminal. VT52 compatible escape sequence commands should not be submitted to the SBD VT100 emulator.

Split Baud Rate

The host serial line baud rate and data format are programmable by means of SETUP (see 2.3). The Transmit and Receive baud rates must be the same.

9.3.2 VT100 Variations

The following VT100 facilities provided by the SBD VT100 emulator vary from the DEC VT100 terminal implementation.

VT100 Set-up Mode

The SBD VT100 emulator provides a Terminal Setup facility similar in principle to the DEC implementation of the VT100 SET-UP facility. Terminal Setup is fully described in Section 9.5. Some of the less frequently changed options are implemented by SETUP (see 2.3).

VT100 Answerback Message

The DEC VT100 implementation allows the user to specify the format and content of the VT100 answerback message sent back to the host computer, when <CTRL>BREAK is typed, or ENQ is received by the VT100 terminal.

The SBD VT100 emulator does not provide a facility for user specified answerback messages. Upon typing <CTRL>BREAK, or receiving ENQ, the SBD will send the following answerback message to the host computer :-

SBD nnnnn Xm.nn<CR>

Where :- nnnnn is the SBD board number (e.g. 01035).
Xm.nn is the firmware type and issue (e.g. A2.01).

SBD Screen Pixel Resolution and the Number of VDU Character Columns

DEC VT100 terminals can provide a character display of either 80 columns or 132 columns in width.

The number of VDU columns provided by the SBD VT100 emulator depends on the SBD pixel screen resolution (Table 9.8):

Screen Resolution	Number Of VDU Character Columns Available
768 by 586	80
768 by 574	80
704 by 526	80
640 by 480	80
512 by 512	80
512 by 384	80
480 by 360	80
384 by 293	64
256 by 256	64

Table 9.8 VDU Columns Available at Different Resolutions

Also refer to Section 3.10 for the description of the SET subroutine to obtain these screen resolutions.

Special Graphics Characters

The DEC VT100 implementation allows special graphics drawing characters to be combined to form solid outlines (e.g. table grids) on the display screen.

The SBD VT100 emulator also provides this facility, but the vertical lines drawn to form solid outlines are discontinuous (i.e. slightly dashed). This phenomenon results from the VDU text internal storage format used within the SBD. The vertical gaps (two or more pixel lines, depending on resolution) between characters are not mapped into the SBD VDU display area in order to conserve internal memory space. The basic font is 5 by 7 pixels, plus two lines to allow for descenders. The higher resolutions duplicate the pixel rows to give a better aspect ratio.

9.4 Extensions to VT100 Features

9.4.1 Using the VDU via DMA

The TXT primitive allows the user to pass character strings to the VDU emulation section of the SBD via the DMA interface. Therefore characters can be displayed in the scrolling text store of the SBD at DMA speeds even if the SBD is not linked to the host computer via a serial line. Furthermore, strings sent to the SBD (except via the ASCII serial interface) may contain embedded escape sequences. These escape sequences will be parsed by the SBD as though they were sent via the serial line and will therefore be treated as possible VT100 escape sequences. This means that application programs using the SBD via the DMA interface have full use of the scrolling text region for tabulated data, help text, error messages etc. For more information on the TXT primitive see the detailed primitive description.

9.4.2 VDU Save and Restore

The SBD's dynamic RAM can be configured to use a portion of the archive storage space as a second scrolling text store. The setting aside of the memory required by the second text store is done by the SET primitive. To create two scrolling text areas simply add 200 to the resolution code argument (see the SET command description in 3.10).

The user can now "save" the current VDU area, that is the VDU being displayed, in the second scrolling text area. Once saved the information can be "restored", when required, by copying the information in the second VDU area back to the VDU area used for display. The user also has the option to "swap" the contents of the displayed VDU area and the second VDU area. The "save", "restore" and "swap" functions are invoked by calling the VDU primitive with a single argument, the value of the argument identifying the function required. See the VDU command description in 3.10 for full details.

Example: send the text string

~VDU 0<CR> (7 characters)

Assuming that the ASCII command introducer is set to tilde (~) (see 2.3), and that a second VDU page has been allocated, this string will cause the current VDU page to be saved. After this, values of 1 and -1 for the argument to VDU will cause the saved page to be written back to the visible VDU, or exchanged with it, respectively.

9.4.3 Printer Port Control

This option is only enabled on an SBD which has an inkjet printer connected to its Peripheral Port (either directly or via a PCU). For such configurations, two escape sequences have been provided to enable/disable SBD transparency mode, i.e. setting the SBD to pass all characters received from the host via the serial interface to the printer until transparency mode is disabled. The escape sequence to enbable transparency mode is

<ESC>[5i

and the escape sequence to to disable transparency mode is

<ESC>[4i

These two escape sequences allow the SBD's printer to be used as a host text printer operating at 32 characters per second. Escape sequences to control the printer's facilities can be embedded in the data stream. For full details on using the SBD's colour printer refer to the "Integrex Colourjet Printer" manual.

For configurations without a printer, these escape sequences are ignored.

9.4.4 Serial Command Control

This option is only enabled on an SBD which has a printer connected to its Peripheral Port (either directly or via a PCU). Two escape sequences are provided to enable/disable the execution of graphics commands received via the Host Port (in either ASCII or binary format). The escape sequence to disable serial command execution is:

<ESC>[4j

and the escape sequence to to re-enable serial command execution is:

<ESC>[5j

When disabled, the commands are still parsed, and not sent to the scrolling VDU area, but they have no effect on the pixel display either. Characters which would normally go to the VDU area still do. However, the escape sequences controlling the printer port are ignored while command execution is disabled.

For configurations without a printer, these escape sequences are ignored.

9.5 Terminal Setup - Operation

A Terminal Setup procedure is provided as part of the VT100 terminal emulation. This procedure is independent of the SBD's Configuration Dialogue (SETUP - see 2.3) and deals with the setting of VDU/terminal attributes only (no graphic device attributes). Figure 9.1 shows the layout of the Terminal Setup display.

The Terminal Setup procedure operates in two modes, Mode A and Mode B. During Mode A, those attributes displayed as a column in the central region of the VDU screen may be set/reset by positioning the cursor on the required attribute and toggling the set/reset indicator from 0 to 1 or vice versa. Mode A is selected on entry to Terminal Setup. While in Mode B the operator can set horizontal tab stops by toggling elements in the ruler along the bottom of the VDU. Tab stops are indicated by digits in reverse video.

To enter Terminal Setup, press the <SET UP> key on the SBD's keyboard (only users with Supervisor keyboards will have access to Terminal Setup). While the SBD is in Terminal Setup, host serial line communication is held up by an XOFF or DTR low signal. On exit from Terminal Setup, the host serial line is re-enabled by sending XON or via DTR.

If the SBD has been SET with two VDU areas (see 9.4.2) then the VDU contents (when the SET UP key was pressed) will be saved in the backup area and restored when Terminal Setup is terminated. If only one scrolling text region exists then the VDU area is cleared when Terminal Setup exits. Thus it is possible to configure the SBD so that Terminal Setup is a non-destructive operation, but only at the expense of archive storage space. See the description of the SET command in 3.10.

On entry to Terminal Setup the SBD sets the screen into VDU mode (no pixel data visible). On exit, the SBD remains in VDU mode.

While in Terminal Setup a subset of the Supervisor's keyboard keys are active. They are given in Table 9.9 below.

Key	Mode A	Mode B
Up arrow) <ESC>) q)	Move up option list, wrapping around from top option to bottom option.	No effect.
Down arrow) <LINE FEED>) r)	Move down option list, wrapping around from bottom to top option.	No effect.
Right arrow) <SPACE>)	No effect.	Move one position to the right on the tab line.
Left arrow) <BACK SPACE>) <DELETE>)	No effect.	Move one position to the left on the tab line.
<TAB>	No effect.	Move right to the next tab stop on the tab line.
<2/@>	No effect.	Toggle tab stop on/off.
<3/#>	No effect.	Clear all tab stops.
<4/$>	Toggle online / local, indicated by LED.	Toggle the online/local status indicated by LED.
<5/%>	Enter Mode B and place cursor on col 1 of tab ruler.	Enter Mode A and place cursor on first option in list.
<6/^>	Toggle option 1/0.	Toggle tab stop on/off.
<0/}>	Reset the SBD.	Reset the SBD.
<SHIFT>S	Save terminal settings in NVRAM and ring bell.	Save terminal settings in NVRAM and ring bell.
<SHIFT>R	Restore the terminal settings from NVRAM.	Restore the terminal settings from NVRAM.
<SET UP>	Exit terminal setup.	Exit terminal setup.

Table 9.9 Active keys in Terminal Setup

Note: keys 7, 8 and 9 have no effect in either mode of Terminal Setup. To alter the line characteristics, use SETUP (see 2.3).

```
TERMINAL SETUP

     Autowrap    1  On-1 Off-0
     Keyclick    0
     Newline     0
     Margin Bell 0

     BLK-1 U/L-0 1
     UK-1  US-0  0




      1         2         3         4         5         6         7         8
2345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Figure 9.1 Appearance of Terminal Setup Menu

Note: on some resolutions on the low-resolution SBD, only 64 tab positions appear.

9.6 Summary of Escape Sequences

This section summarises the escape control sequences supported by the SBD VT100 emulator. The <ESC> character is shown here without angle brackets.

CURSOR MOVEMENT COMMANDS

Function	Sequence
Cursor up	ESC[PnA
Cursor down	ESC[PnB
Cursor forward (right)	ESC[PnC
Cursor backward (left)	ESC[PnD
Direct cursor addressing	ESC[Pl;PcH
Direct cursor addressing	ESC[Pl;Pcf
Index	ESCD
Next line	ESCE
Reverse index	ESCM
Save cursor and attributes	ESC7
Restore cursor attributes	ESC8

CHARACTER ATTRIBUTES

Function	Sequence
Attribute select	ESC[Ps;...;Psm

Ps - 0 or none All attributes off
4 Underscore on
7 Reverse video on

ERASING

Function	Sequence
From cursor to end of line	ESC[K
From cursor to end of line	ESC[0K
From beginning of line to cursor	ESC[1K
Entire line containing cursor	ESC[2K
From cursor to end of screen	ESC[J
From cursor to end of screen	ESC[0J
From beginning of screen to cursor	ESC[1J
Entire screen	ESC[2J

PROGRAMMABLE LEDs

Function	Sequence
Program SBD keyboard LEDs	ESC[Ps;Ps;...Psq

Ps - 0 or none All LEDs off
1 LED L1 on
2 LED L2 on
3 LED L3 on
4 LED L4 on

CHARACTER SETS (G0 AND G1 DESIGNATORS)

Character Set	G0 Designator	G1 Designator
United Kingdom(UK)	ESC(A	ESC)A
United States (USASCII)	ESC(B	ESC)B
Special graphics chars.	ESC(0	ESC)0

SCROLLING REGION

Function	Sequence
Set scrolling region	ESC[Pt;Pbr

Pt - line number of top VDU line in scrolling region.

Pb - line number of bottom VDU line in scrolling region.

TAB STOPS

Function	Sequence
Set tab at active cursor column	ESCH
Clear tab at active cursor column	ESC[g
Clear tab at active cursor column	ESC[0g
Clear all tab positions	ESC[3g

MODES

Mode Name	To Set Mode	To Set Sequence	To Reset Mode	To Reset Sequence
Line feed/new line	New Line	ESC[20h	Line feed	ESC[20l (el)
Cursor key mode	Application	ESC[?1h	Cursor	ESC[?1l (el)
Wraparound	On	ESC[?7h	Off	ESC[?7l (el)
Auto repeat	On	ESC[?8h	Off	ESC[?8l (el)
Keypad mode	Application	ESC=	Numeric	ESC>

REPORTS

Function	Sequence
Cursor position report, invoked by	ESC[6n
Response from SBD is	ESC[Pl;PcR (P el;PcR)
Status report, invoked by	ESC[5n
Response from SBD is	ESC[On (termianl OK)
What are you ? ; invoked by	ESC[c
What are you ? ; invoked by	ESC[0c
Response from SBD is	ESC[?1;0c (base VT100 no options)

APPENDIX A - CONTENTS

Tables

A Error codes and Error Messages

This Appendix (Table A.1) defines all error numbers associated with the SBD and its host support software, together with any error message text as defined in the RSX/RT-11 version of GGS.

The error codes given here are those which can be returned by calls to routines in the GGS host library (not GGSF77).

Key to Table A.1

Error codes are given to base 8 (Oct), base 10 (Dec), and base 16 (Hex).

The severity code (Sev) is given as:

S = success
I = informational
W = warning
E = error
F = severe error

The error source (Src) is given as:

SBD = the SBD itself
Lib = the GGS host library
Drv = the SB: DMA driver

Text is given, where possible, exactly as defined in the GGS library (when configured for full error messages).

Where an error has a specific relationship with one or more commands, these are shown at the extreme right. If an error can occur from more than 4 different commands, these are not shown; just the words "several" or "any".

Table A.1 Error Messages

Oct	Dec	Hex	Sev	Src	Text	Command(s)
0	0	0			<not used>	
1	1	1	S	Lib	<none - success>	any
2	2	2			<not used>	
3	3	3	F	SBD	Illegal number of arguments	any
4	4	4	E	SBD	Bad resolution code	SET
5	5	5	E	SBD	Too many pages	SET
6	6	6	E	SBD	Undefined channel	SEL
7	7	7	E	SBD	Bad colour	SEL
10	8	8	E	SBD	Invalid channel number	DEF SEL
11	9	9	E	SBD	Invalid channel width	DEF
12	10	A	E	SBD	Incompatible width and group	DEF
13	11	B	E	SBD	Invalid mode	SOM
14	12	C	E	SBD	Too many lines	MIX
15	13	D	I	SBD	Coordinate out of range	several
16	14	E	F	SBD	Invalid function code	any
17	15	F	E	SBD	Invalid font code	SCA
20	16	10	E	SBD	Invalid orientation code	SSA SCA
21	17	11	E	SBD	Invalid or incompatible cell height	SSA SCA
22	18	12	E	SBD	Invalid or incompatible cell width	SSA SCA
23	19	13			<not used>	
24	20	14	E	SBD	Invalid archive code	RAR BAR
25	21	15	E	SBD	Archive already open	BAR
26	22	16	E	SBD	Archive not found	RAR DAR
27	23	17	E	SBD	Insufficient archive memory	any
30	24	18	E	SBD	Archive nesting level too deep	RAR
31	25	19	E	SBD	Archiving not active	EAR
32	26	1A	E	SBD	Command illegal when archiving	several
33	27	1B	E	SBD	Archive directory full	BAR
34	28	1C	F	SBD	Cannot recall open archive	RAR
35	29	1D	E	SBD	Bad line type	SLT
36	30	1E	E	SBD	Invalid character	SYM
37	31	1F	I	SBD	Shape too complicated	FLO

Table A.1 Error Messages (continued)

Oct	Dec	Hex	Sev	Src	Text	Command(s)
40	32	20	E	SBD	Bad radius value	CIR CIF ELI ELF
41	33	21	E	SBD	Seed pixel is background colour	FLO
42	34	22	E	SBD	Peripheral not connected	CUR SDM DMP TAB
43	35	23	E	SBD	Tablet format unknown	TAB
44	36	24	E	SBD	Peripheral not ready	SDM DMP TAB
45	37	25	E	SBD	Magnification factor too large	MAG
46	38	26	E	SBD	Coordinate invalid	several
47	39	27	I	SBD	Overlapping areas	ZOO
50	40	28	E	SBD	Cannot restore, nothing saved	RPC VDU
51	41	29	I	SBD	Existing context(s) deleted	SET IDI
52	42	2A	E	SBD	Invalid argument	VDU
53	43	2B	F	SBD	Invalid command block	any
54	44	2C			<not used>	
55	45	2D	W	SBD	Peripheral not active	TRO
56	46	2E			<not used>	
57	47	2F			<not used>	
60	48	30			<not used>	
61	49	31			<not used>	
62	50	32			<not used>	
63	51	33			<not used>	
64	52	34			<not used>	
65	53	35			<not used>	
66	54	36			<not used>	
67	55	37			<not used>	
70	56	38			<not used>	
71	57	39			<not used>	
72	58	3A			<not used>	
73	59	3B			<not used>	
74	60	3C			<not used>	
75	61	3D	W	SBD	Packet sequence error	Ethernet
76	62	3E	F	SBD	Connected to other node	Ethernet
77	63	3F	I	SBD	Aborted (no message text)	

Table A.1 Error Messages (continued)

Oct	Dec	Hex	Sev	Src	Text	Command(s)
100	64	40	E	Lib	Invalid number of arguments	any
101	65	41	E	Lib	Command illegal while blocking	several
102	66	42	W	Lib	Overflow in command block	any
103	67	43	E	Lib	No device active (INI missing)	any
104	68	44	E	Lib	Command block is locked for I/O	EBK EGB
105	69	45	W	Lib	Not blocking	EBK
106	70	46	W	Lib	Block size mismatch	EGB
107	71	47	E	Lib	Device already active	INI
110	72	48	E	Lib	Argument out of range	INI
111	73	49			<not used>	
112	74	4A			<not used>	
113	75	4B	E	Lib	Invalid or reserved event flag	INI
114	76	4C	E	Lib	Invalid or reserved LUN	INI
115	77	4D	E	Lib	Invalid physical unit number	INI
116	78	4E	E	Lib	DMA interface not supported	INI
117	79	4F	E	Lib	Serial ASCII interface not supported	INI
120	80	50	E	Lib	Serial binary interface not supported	INI
121	81	51	E	Lib	<not used>	
122	82	52	E	Lib	No serial interface supported	INI
123	83	53	E	Lib	No command block terminator found	EGB
124	84	54	E	Lib	Image too large - max 131032 pixels	WIB RIB
125	85	55	E	Lib	Bin protocol has no imaging support	WIB RIB
126	86	56	W	Lib	Imaging routine not valid in ASCII	WIB RIB
127	87	57	E	Lib	<not used>	
130	88	58	E	Lib	Invalid Hollerith size	several
131	89	59	E	Lib	Bad argument list	OUTPUT
132	90	5A	E	Lib	Invalid ASCII introducer	CHI
133	91	5B	E	Lib	File access not supported	INI
134	92	5C	F	Lib	Error in filename	INI
135	93	5D			<not used>	
136	94	5E			<not used>	
137	95	5F			<not used>	

Note: codes 140 to 177 (octal) not used.

Table A.1 Error Messages (continued)

Oct	Dec	Hex	Sev	Src	Text
200	-128	80	F	Drv	SB: I/O channel closed
201	-127	81	F	Drv	SB: .SPFUN invalid function code
202	-126	82	F	Lib	Invalid unit number
203	-125	83	F	Drv	SB: device not ready
204	-124	84	F	Lib	I/O channel not open
205	-123	85	F	Lib	H/W error from previous I/O on channel
206	-122	86	F	Lib	Cannot allocate I/O channel
207	-123	87	F	Lib	I/O channel in use or handler not loaded
210	-122	88	F	Drv	Operation aborted or timed out
211	-121	89	F	Drv	Command block not word-aligned
212	-120	8A	F	Lib	No character input available from SBD
213	-119	8B	F	Lib	Non-existent LUN number specified
214	-118	8C	F	Lib	SBD Unit already attached to another program
215	-117	8D	F	Lib	RT11XM - Prog Req data structure not mapped
216	-116	8E	F	Lib	Invalid unit number; LUN not attached.
217	-115	8F	F	Lib	No room in multiterminal output buffer
220	-114	90	F	Lib	Invalid request; function code out of range
221	-113	91			<not used>
222	-112	92			<not used>
223	-111	93			<not used>
224	-110	94			<not used>
225	-109	95			<not used>
226	-108	96			<not used>
227	-107	97			<not used>
230	-106	98			<not used>
231	-105	99			<not used>
232	-104	9A			<not used>
233	-103	9B			<not used>
234	-102	9C			<not used>
235	-101	9D			<not used>
236	-100	9E			<not used>
237	-99	9F			<not used>

The errors above (200 to 237 octal) are produced by the RT-11 host software. They are all fatal. Codes 240 to 377 are not used by the RT-11 library or driver.

Note: codes generated under RSX-11M in the range 200 to 377 (octal) (-128 to -1 decimal) are standard RSX I/O status codes, and should be looked up in the RSX-11M/M-PLUS Executive Reference Manual or the I/O Drivers Manual. They are all considered fatal.

Under VAX/VMS, all errors resulting from system service calls report the standard VMS error codes. They are all considered fatal.

APPENDIX B - CONTENTS

B Example Program Listings

In this Appendix, all listings are given in FORTRAN. They should be readily translateable to other programming languages.

B.1 Generation of archive directory listing

```
C
C       SBDDIR - program to generate archive directory listing
C       RSX-11M or RT-11
C
        PROGRAM SBDDIR
        IMPLICIT INTEGER (A-Z)
        DIMENSION FILE(4),SIZE(4)
C
C       Get SBD number (DMA) e.g. 0 = SB0:
C
        TYPE 100
100     FORMAT('$SBD NUMBER : ')
        ACCEPT *,NUM
C
C       Initialise.  If using VMS use logical name, e.g.
C       CALL INI (,'ggs_dma',2,17) where ggs_dma is a logical name
C       such as DEFINE ggs_dma SBA0
C
        CALL INI(1,NUM,2,17)
C
C       Get line 0 of archive directory and print header
C
        L=0
        CALL DDI(L,LINES,ENTRYS,DUMMY,DUMMY,TSIZE,USED,FREE,DUMMY)
        TYPE 200,NUM
200     FORMAT(' Archive directory listing for SB',I1,':'/)
C
C       Initialise counters
C
        I=1
        LCNT=1
        ACCUM=0
C
C       "WHILE" loop to get next line of 4 entries
C
300     X=I
        CALL DDI(X,FILE(1),SIZE(1),FILE(2),SIZE(2),
     1  FILE(3),SIZE(3),FILE(4),SIZE(4))
        IF (X.EQ.-1) GOTO 700
```

```
C
C       Loop to check all 4 entries and print them on a line.
C
        NUM = 0
        DO 400 ENTRY=1,4
        IF ((FILE(ENTRY).EQ.-1).AND.(SIZE(ENTRY).EQ.-1)) GOTO 400
        NUM = NUM + 1
        ACCUM=ACCUM+SIZE(ENTRY)
400     CONTINUE
        IF (NUM.EQ.0) GOTO 600
        TYPE 500,(FILE(J),SIZE(J),J=1,NUM)
500     FORMAT(' ',4(I7,I5,'.'))
C
C       Increment counts and check whether finished.
C
600     LCNT=LCNT+1
700     I=I+1
        IF(LCNT.LE.LINES) GOTO 300
C
C       Print footer info.
C
        IACC=(ACCUM+255)/256
        TYPE 800,IACC,USED,TSIZE,TSIZE-(IACC+FREE),ENTRYS,FREE
800     FORMAT(1X,/,' Total of ',I4,'./',I4,'. blocks used out of ',
     1  I4,'.',/,' Size of directory = ',I4,'. blocks, ',I5,
     2  '. archive records',/,' Free blocks = ',I4,'.')
        STOP
        END
```

B.2 Program to demonstrate nested archiving

```
C ---
C     CROWD - draw faces (RSX-11M/RT-11)
C
        PROGRAM CROWD
        IMPLICIT INTEGER(A-Z)
        REAL RAN
C
C     Initialise DMA device SB0:
C     Suppress I and W severity errors.
C
        STATUS=INI(,0,,3089)
        IF (STATUS.EQ.1) GOTO 10
        TYPE 100
100     FORMAT(' Initialisation failure')
        GOTO 999
C
C    Define archive numbers - should be unique to this program.
C
10      FACE = 2001
        EYE = 2002
        PIMPLE = 2003
C
C    Create archive for pimple.
C    Use subroutine CREATE (below) to minimise code repetition.
C
        CALL CREATE(PIMPLE)
        CALL CIF(0,0,1)
        CALL EAR
C
C     Create archive for eye
C
        CALL CREATE(EYE)
        CALL ELI(0,0,10,6)
        CALL CIR(0,0,4)
        CALL CIR(0,0,2)
        CALL EAR
C
C     Create face archive, including mouth, nose, eyes and pimples
C     Note eyes and pimples are offset sub-archives, LIX here acts
C     relatively to index registers at start of archive
C
        CALL CREATE(FACE)
        CALL ELI(0,0,30,50)
        CALL BOX(-10,10,10,15)
        CALL VEC(0,-6,4,2)
        CALL VEC(-4,2)
        CALL VEC(0,-6)
        CALL LIX(-15,-15)
        CALL RAR(EYE)
        CALL LIX(15,-15)
```

```
          CALL RAR(EYE)
          CALL LIX(-20,0)
          CALL RAR(PIMPLE)
          CALL LIX(20,0)
          CALL RAR(PIMPLE)
          CALL LIX(-16,-5)
          CALL RAR(PIMPLE)
          CALL LIX(16,-3)
          CALL RAR(PIMPLE)
          CALL LIX(18,-4)
          CALL RAR(PIMPLE)
          CALL LIX(-13,2)
          CALL RAR(PIMPLE)
          CALL LIX(-25,5)
          CALL RAR(PIMPLE)
          CALL EAR
C
C     Perform usual initialisations
C
          CALL SEL(0,1,0)
          CALL SAP(0)
          CALL PAG(0)
          CALL LIX(0,0)
          CALL MIX(0)
          CALL RUB
C
C     Loop to draw faces all over screen
C     LIX here acts absolutely
C
          SEED1=0
          SEED2=0
          DO 50 I=1,1000
          POSX=IFIX(RAN(SEED1,SEED2)*767.0)
          POSY=IFIX(RAN(SEED1,SEED2)*573.0)
          COLOUR=IFIX(RAN(SEED1,SEED2)*15.0)
          CALL SEL(0,COLOUR,0)
          CALL LIX(POSX,POSY)
          CALL RAR(FACE)
50        CONTINUE
C
C     Tidy up and exit
C
999       CALL LIX(0,0)
          STOP
          END
```

```
C
C    Subroutine CREATE.
C    Opens specified archive - if it already exists, deletes it
C    and creates it again.
C
         SUBROUTINE CREATE (NUM)
         IMPLICIT INTEGER (A-Z)
         STATUS=BAR(NUM)
         IF (STATUS.EQ.1) GOTO 799
         IF (STATUS.NE.21) GOTO 700
         STATUS=DAR(NUM)
         IF (STATUS.NE.1) GOTO 700
         STATUS=BAR(NUM)
         IF (STATUS.EQ.1) GOTO 799
700      TYPE 710,NUM,STATUS
710      FORMAT(' Failure in archive no. ',I6, - Status = ',I3)
         CALL LIX(0,0)
         STOP
799      RETURN
         END
```

B.3 Subroutine to map an arbitrary rectangle to screen

```
C
C     Uses TIN and TAB routines to map a rectangular surface at
C     any angle and position relative to the tablet coordinate
C     system origin, to the whole of the SBD's screen.  Subsequent
C     calls to TAB with the puck inside the defined rectangle would
C     return coordinates inside the screen bounds.  Note that the
C     scaling operation is independent of the resolution of the
C     tablet in use.  Assumes that calling program has called INI.
C
C     The surface is mapped to the screen using the following procedure:
C
C     (X1,Y1) +------------------+
C             |                  |
C             |                  |
C             |                  |
C             |                  |
C     (X2,Y2) +------------------+ (X3,Y3)
C
C     1. Points (X1,Y1), (X2,Y2) and (X3,Y3) are digitised using
C        unmapped calls to TAB.
C
C     2. The angle of rotation (anticlockwise) of the base of the
C        rectangle from the tablet X direction is computed using
C        simple trigonometry, as COSROT and SINROT.
C
C     3. The sides of the rectangle are computed using Pythagoras'
C        Theorem and mapped to the screen size (i.e. X and Y scale
C        factors (XF,YF)).
C
C     4. The shift of the rectangle origin from the tablet origin
C        is computed using the transformations:
C
C        XS = (X2*COSROT + Y2*SINROT) * XF
C        YS = (Y2*COSROT - X2*SINROT) * YF
C
C     5. The calculated information is used in a TIN call to set up
C        the tablet-to-screen mapping, independent of the tablet
C        resolution.
C
        SUBROUTINE TABMAP
        IMPLICIT INTEGER (A-Z)
        REAL SQRT
        REAL RWIDTH,RHEIGH
        REAL XF,YF,XS,YS,SINROT,COSROT
100     TYPE 200
200     FORMAT('$Enter Screen Size (X,Y) : ')
        READ(5,*,END=9999) IX,IY
C
C     Calibrate tablet/table surface.
C
```

```
      ID = 0    ! pass raw coordinates
      IW = 1    ! wait for button release
      IC = 0    ! do-not-display cursor
C
      TYPE 300
300   FORMAT (' Digitise the TOP LEFT hand corner of surface')
      CALL TAB(IX1,IY1,ID,IB,IW,IC)
C
      TYPE 400
400   FORMAT (' Digitise the BOTTOM LEFT hand corner of surface')
      CALL TAB(IX2,IY2,ID,IB,IW,IC)
C
      TYPE 500
500   FORMAT (' Digitise the BOTTOM RIGHT hand corner of surface')
      CALL TAB(IX3,IY3,ID,IB,IW,IC)
C
C     Compute unscaled lengths of edges of rectangle
C
      RWIDTH = SQRT((IX3-IX2)**2 + (IY3-IY2)**2)
      RHEIGH = SQRT((IX1-IX2)**2 + (IY1-IY2)**2)
C
C     Compute sine and cosine of rotational angle
C
      SINROT = (IY3-IY2)/RWIDTH
      COSROT = (IX3-IX2)/RWIDTH
C
C     Compute X and Y scale factors
C
      XF = IX/RWIDTH
      YF = IY/RHEIGH
C
C     Compute origin shifts
C
      XS = (IX2*COSROT + IY2*SINROT) * XF
      YS = (IY2*COSROT - IX2*SINROT) * XF
C
C     Set up tablet to screen mapping
C
      CALL TIN(XF,YF,XS,YS,SINROT,COSROT)
C
C     Return to main program
C
      RETURN
9999  STOP
      END
```

APPENDIX C - CONTENTS

C Alphabetic Host Library Subroutine Summary

This appendix presents all forms of all the commands available under the GGS host subroutine library. It may also be used as a general reference to the command set of the SBD. Refer to 3.10 for detailed descriptions of argument ranges and meanings.

The commands are given in a pseudo-high-level-language call format, with the command mnemonic followed by the arguments in round brackets. Optional arguments are enclosed in square brackets (see 3.8). Arguments are given explanatory English-type names to aid clarity - this may cause some command descriptions to be too long for one line, so commands may be split over more than one line using a hyphen continuation indicator, viz.:

```
VEC (X_start.on, Y_start.on,-
     X_end.on, Y_end.on)
```

Argument qualifiers are given after a full stop (see example above), and have the following meaning:

```
o = output argument (host to SBD)
i = input argument (SBD to host)
b = output argument, overwritten on input

n = integer (word on PDP-11. On VAX, may be word or longword if
    output only, must be longword if input)
f = floating-point (32-bit DEC F_floating)
a = address of word array
b = address of longword array (VAX), word array (PDP)
s = null-terminated string (PDP), string descriptor (VAX)
h = byte or character array
```

C.1 Command Descriptions

ALP -- pixel text

```
ALP (string.os)
ALP (char_count.on, string.oh)
ALP (X_start.on, Y_start.on, string.os)
ALP (X_start.on, Y_start.on, char_count.on, string.oh)
```

BAR -- begin archiving

```
BAR (archive_number.on)
```

BBK -- begin blocking

```
BBK (array.oa, size_in_words.on)
```

BIT -- draw pixel

```
BIT (X_pos.on, Y_pos.on)
```

BLK -- draw solid rectangle

```
BLK (X_corner2.on, Y_corner2.on)
BLK (X_corner1.on, Y_corner1.on, X_corner2.on, Y_corner2.on)
```

BOX -- draw hollow rectangle

```
BOX (X_corner2.on, Y_corner2.on)
BOX (X_corner1.on, Y_corner1.on, X_corner2.on, Y_corner2.on)
```

CHI -- change ASCII introducer

```
CHI (character.os)
```

CIF -- draw solid circle

```
CIF (X_centre.on, Y_centre.on, radius.on)
CIF (X_centre.on, Y_centre.on, radius.on, octant_code.on)
```

CIR -- draw hollow circle

```
CIR (X_centre.on, Y_centre.on, radius.on)
CIR (X_centre.on, Y_centre.on, radius.on, octant_code.on)
```

CUR -- activate pixel cursor

```
CUR (X_start_&_finish.bn, Y_start_&_finish.bn)
CUR (X_start_&_finish.bn, Y_start_&_finish.bn, transmit_key.in)
CUR (X_start_&_finish.bn, Y_start_&_finish.bn,-
     box_width.on, box_height.on)
CUR (X_start_&_finish.bn, Y_start_&_finish.bn, transmit_key.in,
     box_width.on, box_height.on)
```

DAR -- delete archive

```
DAR (archive_number.on)
```

DDI -- get archive directory information

```
DDI (line.bn,-
     archive_num1.in, archive_size1.in,-
     archive_num2.in, archive_size2.in,-
     archive_num3.in, archive_size3.in,-
     archive_num4.in, archive_size4.in)
```

DEF -- define channel

```
DEF (channel.on, mode.on, no_of_planes.on, start_plane.on)
```

DLT -- define line type

```
DLT (bit_pattern.on)
```

DMP -- print screen on inkjet printer

```
DMP
DMP (timeout_ticks.on)
DMP (timeout_ticks.on, mode_flag.on)
DMP (timeout_ticks.on, mode_flag.on,-
     X_corner1.on, Y_corner1.on, X_corner2.on, Y_corner2.on)
```

EAR -- end archiving

```
EAR
EAR (size_in_words.in)
```

EBK -- end blocking

```
EBK
EBK (size_in_words.in)
```

EGB -- execute block

```
EGB (command_block.oa, size_in_words.on)
```

ELF -- solid ellipse

```
ELF (X_centre.on, Y_centre.on, X_radius.on, Y_radius.on)
ELF (X_centre.on, Y_centre.on, X_radius.on, Y_radius.on,-
     quadrant_code.on)
```

ELI -- hollow ellipse

```
ELI (X_centre.on, Y_centre.on, X_radius.on, Y_radius.on)
ELI (X_centre.on, Y_centre.on, X_radius.on, Y_radius.on,-
     quadrant_code.on)
```

FIN --close device

```
FIN
```

FLO -- fill area

```
FLO (X_seed.on, Y_seed.on)
```

FSH -- set flashing (Model B)

```
FSH (LUT_1_ticks.on, LUT_2_ticks.on)
```

IDI -- initialise archive directory

```
IDI
```

INI -- initialise library

Since INI is operating-system-dependent, it cannot be summarised here. See 3.6. The RSX-11M format is:

```
INI ([logical_unit_number.on],[physical_unit_number.on],-
     [event_flag.on],[mode.on])
```

VAX/VMS formats are:

```
INI (,[device_name.os],[event_flag.on],[mode.on])
INI (dev_name_length.on,device_name.oh,[event_flag.on],[mode.on])
```

LIX -- load index registers

```
LIX (X_offset.on, Y_offset.on)
```

MAG -- magnified pixel text

```
MAG (magnification.on, string.os)
MAG (magnification.on, char_count.on, string.oh)
MAG (X_cell.on, Y_cell.on, magnification.on, string.os)
MAG (X_cell.on, Y_cell.on, magnification.on,-
     char_count.on, string.oh)
```

MAT -- load mapping tables (Model B)

```
MAT (colour_values.oa)
MAT (LUT_number2.on, VDU_colour_value.on)
```

MES -- set error mode

```
MES
MES (flag_bits.on)
```

MIX -- set pixel/VDU mix

```
MIX
MIX (VDU_lines.on)
```

PAG -- set display page

```
PAG (page_number.on)
```

PIX -- select pixel data only

```
PIX
```

RAR -- recall archive

```
RAR (archive_number.on)
```

RIB -- read image block

```
RIB (X_corner1.on, Y_corner1.on, X_corner2.on, Y_corner2.on,-
     pixels_per_word.on, pixel_array.ia)
```

RPC -- restore previous context

```
RPC
```

RUB -- erase rectangle

```
RUB
RUB (X_corner1.on, Y_corner1.on, X_corner2.on, Y_corner2.on)
```

SAP -- select access page

```
SAP (page_number.on)
```

SAS -- save context in archive space

```
SAS
```

SCA -- set character attributes

```
SCA (cell_width.on, cell_height.on, orientation.on, font.on)
```

SDM -- set print dump mapping

```
SDM (mapping_table.oa)
```

SEL -- select channel

```
SEL (channel.on, foreground_colour.on, background_colour.on)
SEL (foreground_channel.on, foreground_colour.on,-
     background_colour.on, background_channel.on)
```

SET -- set resolution and pages

```
SET (resolution_code.on)
SET (resolution_code.on, number_of_pages.on)
```

SLT -- select line type

```
SLT (line_type_code.on)
```

SOM -- select output mode

```
SOM (output_mode.on)
```

SSA -- select symbol attributes

```
SSA (orientation.on)
SSA (cell_width.on, cell_height.on, orientation.on)
```

SYM -- define or display symbol

```
SYM (string.os)
SYM (bit_pattern_array.oa, char.os)
SYM (X_cell.on, Y_cell.on, string.os)
SYM (bit_pattern_1.on, [bit_pattern_2.on,...[bit_pattern_14.on,]]-
     char.os)
```

SYN -- synchronise to field flyback

```
SYN
SYN (flag.on)
```

TAB -- activate tablet

```
TAB (X_finish.in, Y_finish.in, transform_flag.bn, button.in,-
     wait_flag.bn, cursor_flag.bn)
TAB (X_finish.in, Y_finish.in, transform_flag.bn, button.in,-
     wait_flag.bn, cursor_flag.bn, retry_flag.on)
```

TIN -- set tablet transformations

```
TIN (X_scale.of, Y_scale.of, X_shift.of, Y_shift.of,-
     sine_rotation.of, cos_rotation.of)
```

TRA -- initiate trackerball sample mode

```
TRA (X_start_or_sample.bn, Y_start_or_sample.bn, flag_in_key_out.bn)
```

TRO -- cancel trackerball sample mode

```
TRO
```

TXT -- text to VDU area

```
TXT (string.os)
```

VDU -- set VDU or save/restore

```
VDU
VDU (save_restore_flag.on)
```

VEC -- draw vector

```
VEC (X_end.on, Y_end.on)
VEC (X_start.on, Y_start.on, X_end.on, Y_end.on)
```

WIB -- write image array

```
WIB (X_corner1.on, Y_corner1.on, X_corner2.on, Y_corner2.on,-
     pixels_per_word.on, pixel_array.oa)
```

ZOO -- copy, zoom or reduce pixel rectangle

```
ZOO
ZOO (magnification.on)
ZOO (source_page.on, destination_page.on)
ZOO (source_X_corner1.on, source_Y_corner1.on,-
     magnification.on)
ZOO (source_X_corner1.on, source_Y_corner1.on,-
     source_X_corner2.on, source_Y_corner2.on)
ZOO (source_X_corner1.on, source_Y_corner1.on,-
     source_page.on, destination_page.on, magnification.on)
ZOO (source_X_corner1.on, source_Y_corner1.on,-
     source_X_corner2.on, source_Y_corner2.on,-
     source_page.on, destination_page.on)
ZOO (source_X_corner1.on, source_Y_corner1.on,-
     source_X_corner2.on, source_Y_corner2.on,-
     destination_X_corner1.on, destination_Y_corner1.on,-
     magnification.on)
ZOO (source_X_corner1.on, source_Y_corner1.on,-
     source_X_corner2.on, source_Y_corner2.on,-
     destination_X_corner1.on, destination_Y_corner1.on,-
     destination_X_corner2.on, destination_Y_corner2.on)
ZOO (source_X_corner1.on, source_Y_corner1.on,-
     source_X_corner2.on, source_Y_corner2.on,-
     destination_X_corner1.on, destination_Y_corner1.on,-
     source_page.on, destination_page.on, magnification.on)
ZOO (source_X_corner1.on, source_Y_corner1.on,-
     source_X_corner2.on, source_Y_corner2.on,-
     destination_X_corner1.on, destination_Y_corner1.on,-
     destination_X_corner2.on,-destination_Y_corner2.on,-
     source_page.on, destination_page.on)
```

Supplement VER - CONTENTS

Supplement VER - Firmware Version Update Descriptions

VER.1 Introduction

This Supplement describes the functional differences (in the user interface) between the significant releases of SBD firmware. It is not intended as a complete description of all changes, but as a guide to enable the user to determine where an earlier release of firmware may differ from that described in this manual. The releases defined as significant are:

Versn	Release Date	Purpose of release
1.2q	03-Oct-84	First functional release - supported SIT serial terminals.
1.7	25-Mar-85	First full release - incorporating all functions defined in product literature. All customers were offered a free upgrade to this standard.
1.10	03-Jul-85	Interim release of enhancements and corrections.
2.1	12-Nov-85	Model B support.

All other releases were considered temporary and were designed to solve specific problems.

Differences between these four significant releases are given below in the form of a table. For users of SITs and printer boards with 1.2q firmware, a command summary is given in VER.2 which explains the differences between 1.2q and 2.1 (described in this manual). The reference numbers quoted are internal to PPL. The items are classified according to type (E = enhancement, C = correction) and functional area as follows:

Ser Host - Serial interface to host
Ser Periph - Serial interface to peripheral
DMA - DMA interface
Primitives - Graphics commands
VT100 - VT100 emulation
SETUP - Configuration dialogue
Powerup - Power-up diagnostics
Hardware - Support for new versions of SBD
Screen - Screen formatting and control
Arcaid - Archiving option

Differences between V1.2q and V1.7

Ref No(s)	Details	Functional Area	Typ	Rel
8	Support for EIA modem control signals CTS/DTR	Ser Host	E	1.7
10	Allow ignoring of parity bit	Ser Host	E	1.7
51	Correct 1200 baud line rate	Ser Host	C	1.7
52	Remove <LF> char from return status/values	Ser Host	C	1.7
56	Allow disabling of ASCII serial introducer	Ser Host	E	1.7
58	Success return as 00<CR> rather than 4 nulls	Ser Host	E	1.7
61	Allow selection of any parity type	Ser Host	E	1.7
73	Allow selection of 7 or 8 data bits	Ser Host	E	1.7
75	Allow selection of 1 or 2 stop bits	Ser Host	E	1.7
15	Improve reliability of communication between SBD and PCU	Ser Periph	E	1.7
9	CHI command to allow changing of ASCII introducer character	Primitives	E	1.7
14	TIN and TAB commands for control of graphics tablet	Primitives	E	1.7
18	WIB and RIB commands for image transfer (DMA)	Primitives	E	1.7
19	ZOO command for raster copy/zoom	Primitives	E	1.7
25	TXT command for text to VDU area	Primitives	E	1.7
36	CIF command for solid circles	Primitives	E	1.7
37	ELF command for solid ellipses	Primitives	E	1.7
41	RST command (reset, for test purposes)	Primitives	E	1.7
44	SAS and RPC commands for context switching	Primitives	E	1.7

Ref No(s)	Details	Functional Area	Typ	Rel
22	ALP/MAG: correct positioning of descender characters	Primitives	C	1.7
67	ALP - return correct DMA offset to error	Primitives	C	1.7
72	ALP/MAG: origin of text string should always be the corner nearest (0,0) - only affects orientation 3.	Primitives	E	1.7
76	MAG: implement orientations 1 to 3	Primitives	E	1.7
24	DMP: implement 8-colour fast dump	Primitives	E	1.7
49	DMP: incorrect when Y axis inverted	Primitives	C	1.7
45	Implement second VDU page (SET/VDU)	Primitives	E	1.7
55	SET with one argument now creates max number of pixel pages, and enters PIX mode	Primitives	E	1.7
26	Severity classification (S,I,W,E,F) on errors detected in commands	Primitives	E	1.7
23	Implement VDU tab positions every 8 columns	VT100	E	1.7
43	General VT100 enhancements: programmable tabs, bell, Terminal Setup, UK ASCII	VT100	E	1.7
77	Allow embedded control characters in escape sequences	VT100	E	1.7
17	Implement software protection scheme for options (ARCAID etc).	SETUP	E	1.7
34	Allow power-up in VDU mode (no pixels)	SETUP	E	1.7
68	Allow power-up as 768x574 (interlace) with 2 pages	SETUP	E	1.7
71	Bus error when SETUP entered with CUR active	SETUP	C	1.7
32	Enable keyboard if power-up error occurs	Powerup	C	1.7

Ref No(s)	Details	Functional Area	Typ	Rel
33	Full power-up checking and LED diagnostics	Powerup	E	1.7
20, 42	Support low-resolution version of SBD	Hardware	E	1.7
60, 64	Interlace mode sometimes mixed up frames	Screen	E	1.7

Differences between V1.7 and V1.10

Ref No(s)	Details	Functional Area	Typ	Rel
90	Allow printer to be switched off while connected to SBD	Ser Periph	E	1.10
86	Tighter checks on structure of DMA command blocks	DMA	E	1.10
89	Retry bus errors on DMA registers at power-up	DMA	E	1.10
94	SYN command (synchronise to field flyback)	Primitives	E	1.10
80	DMP: various enhancements including: dump area, orientation control, speed, alternate line prints	Primitives	E	1.10
97	SDM: allow mapping in 8-colour mode	Primitives	E	1.10
40	WIB and RIB implemented in serial binary protocol	Primitives	E	1.10
54, 83	ALP/MAG: reduce flicker on rapidly updated strings (orientations 0 and 1 only)	Primitives	E	1.10
78	VDU cursor could disappear on NO SCRL	VT100	C	1.10
84	Tab character after last tab position could deformat the screen	VT100	C	1.10
88	Implement escape sequences for tab control	VT100	E	1.10
81	Improve SETUP driven from external VDU	SETUP	E	1.10
91	Support high I/O page addresses	Hardware	E	1.10
93	Bus error if more than 128 archives	Arcaid	C	1.10

Differences between V1.10 and V2.1

Ref No(s)	Details	Functional Area	Typ	Rel
115	TRA and TRO commands for background mouse/trackerball control, plus ISI support	Primitives	E	2.1
120	BGO general inquiry primitive (unsupported)	Primitives	E	2.1
104, 134	ZOO: single column copy can cause black rectangle	Primitives	C	2.1
118	ZOO: overlapping areas sometimes not detected	Primitives	C	2.1
119	ZOO: 2x magnification speeded up	Primitives	E	2.1
98	SEL: allow selection of foreground and background channel independently. Also add channels 6 & 7 (2-plane)	Primitives	E	2.1
131	SEL should not change foreground colour if background colour was invalid	Primitives	C	2.1
101	ALP/MAG: Lowercase J does not descend	Primitives	C	2.1
102, 105, 111, 121	SET: improve parameter checking	Primitives	C	2.1
107	SET: improve conformance with manual on initialisations. IDI not to delete symbols.	Primitives	C	2.1
117	SYM: spurious "coordinate out of range" error if close to screen edge	Primitives	C	2.1
123	SYM: improve argument checking	Primitives	C	2.1
125	RUB operates incorrectly with channels 6 & 7	Primitives	C	2.1
126	Ensure correct number of return values still sent if non-fatal error occurs	Primitives	C	2.1
132	CUR: cursor left on screen if peripheral not present	Primitives	C	2.1

Ref No('s)	Details	Functional Area	Typ	Rel
128	Process <ESC>D correctly	VT100	C	2.1
103	Allow MIX(0) on power-up	SETUP	E	2.1
113	DMA address 76xnn0 not correctly stored if x > 0	SETUP	C	2.1
99, 100	Support Model B (MAT, FSH commands; fuse fail detect; watchdog timer)	Hardware	E	2.1
106	Bus error on DAR(-1)	Arcaid	C	2.1
109	Ignore SET if archiving	Arcaid	C	2.1
110, 129	11th level of archive causes crash	Arcaid	C	2.1
136	Improve DDI argument checking	Arcaid	C	2.1

VER.2 Version 1.2q Command Differences

The following section summarises, for each graphics command, the major V1.2q functions which have changed for V2.1 (described in this manual). Supervisor Industrial Terminal (SIT) users, and users of printer boards, with 1.2q firmware should refer to this summary in conjunction with the command descriptions in 3.10. Only those functions which are relevant to SIT/printer board users are described. Only functional differences are noted here; for error corrections see VER.1.

Differences are described here as they appear to the 1.2q user (not the 2.1 user).

ALP - display text on pixel area

The origin of strings is at the upper left corner of the first character (which is not necessarily the point nearest to the upper left of the screen, in some orientations - the corner rotates with the character), rather than at the point closest to the screen origin (0,0). This means that if the Y axis is inverted, string cells will be shifted relative to their positions with normal Y orientation (increasing downward).

All strings are written by erasing the background cells before writing the characters. This means that with long, rapidly updated strings, some flicker is possible (though this is unlikely to be a problem at serial line command speeds).

BGO - general inquiry command

Not implemented.

CHI - change ASCII command introducer

Not implemented. ASCII command introducer must be tilde (~).

CIF - display solid circle

Not implemented.

DMP - print screen on inkjet printer

Only the 16-colour, full-page print mode is allowed (call formats 1 and 2). No speed optimisation is used.

ELF - display solid ellipse

Not implemented.

FLO - fill area

FLO does not support line types. It fills with foreground colour only.

FSH - set flash rate

Not implemented (Model B).

MAG - display magnified string

MAG does not support orientations other than zero.

MAT - load lookup tables

Not implemented (Model B).

RIB - read image block

Not implemented.

RPC - restore context

Not implemented.

RST - reset

Not implemented.

SAS - save context

Not implemented.

SDM - set printer colour mapping

Only the 16-colour mode is supported.

SEL - select channel and colours

You cannot select independent foreground and background channels. Channels 6 and 7 do not exist.

SET - select resolution and pages

SET initialises the screen to white foreground and black background, rather than to the values selected by SETUP.

The second VDU page is not implemented.

SET with one argument creates only 1 pixel page, and enters MIX mode.

SET deletes any open archive when called. No error is reported.

SYN - synchronise to field interrupt

Not implemented.

TAB - activate graphics tablet

Not implemented.

TIN - set tablet mapping

Not implemented.

TRA - activate trackerball/mouse

Not implemented.

TRO - trackerball/mouse off

Not implemented.

TXT - display text on VDU area

Not implemented.

VDU - set VDU area visible

VDU does not support the second VDU page (no arguments allowed).

WIB - write image block

Not implemented.

ZOO - raster copy/scale

Not implemented.

Configuration dialogue (SETUP)

The software enable page is not implemented.

The SBD cannot power up with zero or 24 VDU lines visible. You must call the MIX, PIX or VDU commands after power-up to achieve this.

Serial lines and VDU emulation

Host serial data format must be 8 data bits, no parity, 1 stop bit.

The SBD returns status and return values down the serial line with <CR><LF> appended. Success status (no return values) is returned as four nulls rather than 00<CR><LF>.

You cannot disable or change the ASCII introducer, tilde (~). This means that you cannot edit files with this character in them.

Only XON/XOFF data rate control is supported. EIA modem control signals are not monitored or controlled.

Control characters embedded in escape sequences will cause parsing of the escape sequence to terminate.

No tab positions are available on the VDU.

You cannot disable the execution of serial commands on printer boards.

There is no equivalent to VT100 SETUP for setting tabs, keyclick etc.

The printer will halt the SBD if switched off while connected to the SBD.

Diagnostics

Powerup diagnostic checking is limited to EPROM checksum, NVRAM checksum and peripheral identification. The LED is not used for diagnostic indication. No status message is displayed if SETUP is entered as a result of diagnostic checks.