25th February 1981

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
			6144												

SPECIFICATIONS FOR POLYWOG OPERATING SYSTEM

CONTENTS

1. INTRODUCTION

This document specifies the software for the POLYWOG micro computer. Although the initial evaluation will use the TSC Flex operating system and the associated BASIC interpreter, both an operating system and a BASIC interpreter will be developed for POLYWOG following this phase. The changes and extensions required to be made to both the Flex operating system and the BASIC interpreter for the evaluation are specified in this document.

1.1. CONVENTIONS

Throughout these specifications the following conventions apply:

1. The bits within a byte are numbered from 7 through 0, with bit 0 being the low order bit. (If only bit 0 is on in a byte, the byte has a binary value of 1.)

2. All arrays and screens are numbered with (0,0) being in the top left hand corner, with the first row and column being numbered 0.

3. Reference to a point in arrays is by Row, Column. i.e. the vertical displacement followed by the horizontal displacement.

1.2. SCREEN AREAS

The Polywog has 5 screen areas as follows:

T1	Teletext 24 lines. This may operate in either 80, 40 or 20 characters per line mode. These are referred to as T1(80), T1(40) and T1(20). The user only has access to the first 23 lines, the 24th line being reserved for messages from the operating system.
G1	Graphics. 204 pixel lines. This may operate in either 480 or 240 pixels per line mode. These are referred to as G1(480) and G1(240).
T2	Teletext 24 lines. This screen is fixed at 40 characters per line. This is referred to as T2 or T2(40). The user only has access to the first 23 lines.
G2	Graphics. 204 pixel lines. This screen is fixed at 240 pixels per line. This is referred to as G2 or G2(240).
B	Background. ½ intensity colours

Up to 4 screen areas and the background may be displayed similtaneously but screen T2 cannot be displayed while screen T1(80) is displayed. Similarly screen G2 cannot be displayed while screen G1(480) is being displayed.

The screens have a priority fixed in the order they are listed above. A pixel on any screen will only be displayed if there is no pixel in the same place on any higher priority screen being displayed. The programmer may determine which screens are to be displayed, but cannot alter the priority.

The graphics screens do not access the bottom 4 lines of the teletext screen.

Teletext screens T1(40) and T2 use the teletext conventions with control codes inserted within the text. Up to 7 colours may be displayed on these screens. These may be changed at any time within the display by the insertion of the appropriate code. 80 by 69 chunky graphics may be used.

Reverse video is also supported and may be switched on and off prior to printing to the screen. Reverse video reverses the colours in both the foreground and the background. Characters written while in reverse video mode have bit 7 equal to 1.

T1(20) and T1(80) screens handle teletext in the same way as T1(40) except that colour control codes are ignored. These screens have no background colour, and the foreground colour is restricted to only one colour on the screen at a time. This may be any of red, green, blue or white.

Graphics screens may use any of 4 colours, which may be defined within the screen as required.

1.3. <u>MEMORY</u>

Up to 128K bytes of memory may be attached to each POLYWOG. At any time the memory above 64K may be mapped onto the lower 64K and accessed.

1.4. <u>DISK HANDLING AND COMMUNICATIONS</u>

The POLYWOG system is configured so that up to 16 POLYWOGS share the same set of disks. The disks are controlled by a MASTER POLYWOG without screen or keyboard. The operating system is split, so that the user POLYWOGS contain only those interfaces required to interface with BASIC and to send requests via the communications line, for specific disk transactions to be carried out. The disk POLYWOG contains those parts of the operating required to interface with the disks and carry out the actual disk handling.

1.5. TELETEXT SCREENS

The teletext screens consist of 2 memory areas each of 40 x 24 characters, ie 960 bytes. Each byte in the memory represents a character on the screen. Colour, graphics, flashing and double height fields are controlled by the use of special control characters. Although these each take up one byte in the screen memory, they are not displayed on the screen, but the current colour set for the background is displayed. At the start of each row, all modes are reset to:

In graphics repeats last graphics char.

Colour: White
Background: Black
Flashing: Non Flashing
Height: Normal height
Characters/graphics: Characters

The TELETEXT control characters are:

$01	(1)	Red Characters
$02	(2)	Green Characters
$03	(3)	Yellow Characters
$04	(4)	Blue Characters
$05	(5)	Magenta Characters
$06	(6)	Cyan Characters
$07	(7)	White Characters *
$08	(8)	Flashing ON
$09	(9)	Flashing OFF *
$0C	(12)	Normal Height *
$0D	(13)	Double Height
$11	(17)	Red Graphics
$12	(18)	Green Graphics
$13	(19)	Yellow Graphics
$14	(20)	Blue Graphics
$15	(21)	Magenta Graphics
$16	(22)	Cyan Graphics
$17	(23)	White Graphics *
$19	(25)	Contiguous Graphics *
$1A	(26)	Separated Graphics
$1C	(28)	Black Background *
$1D	(29)	New Background
$1E	(30)	Start Graphics
$1F	(31)	~~Start Characters~~ * Release Graphics

Use only on T1 - which also affects T2 + following line on T2

0E SO
0F SI

* indicates the initial setting on each row.

1.6. GRAPHICS SCREENS

Two graphics screen areas are provided each capable of specifying 240 graphic pixels across the page and 204 down. Alternatively, a single 480 x 204 screen may be used. The graphics may access all but the bottom three and a half lines of the teletext screen.

Each graphics screen is held in an 8K byte area, the user specifying which screen areas are required. If an area is not required, that memory is added to the users program area.

The screen is held in memory as a series of bytes, each byte describing 6 horizontal pixels. Bits 7 and 6 specify the colour of the 6 bits, and Bits 5 to 0 specify which of the 6 pixels are displayed, 1 implying that a pixel is ON and 0 implying that a pixel is OFF. The colour codes used in bits 7 and 6 are:

00	=	BLUE
01	=	GREEN
10	=	RED
11	=	WHITE

2. <u>SOFTWARE IMPLEMENTATION</u>

The specification for the POLYWOG operating software is described under an implementation priority so that each level defined is completely defined prior to definition of the next priority.

Priority 1 must be attained prior to having a POLYWOG available for development of the application software and the capture of diagrams and pictures through use of the digitizer. Priority 1 uses stand alone POLYWOGS with a line printer, discs and RS232 serial interface attached.

Priority 2 must be reached before the evaluation takes place. By this time POLYWOG must be operating with a shared set of disks and printer.

Priority 3 specifies features required before marketing of the POLYWOG on a wider basis can be undertaken. These do not necessarily need to be implemented in the order specified.

3. IMPLEMENTATION - PRIORITY 1

The software in Priority 1 is the software required to set up a stand alone POLYWOG with disks, printer and serial RS232C port attached directly onto the bus.

3.1. OPERATING SYSTEM

The current Flex system requires little modification to operate within a standalone POLYWOG with disks attached directly to it. Due to the need to protect certain areas within the system memory these routines are accessible only through a software interrupt at the machine code level. The form for passing parameters into these routines is that following the software interrupt $3F (63) a single byte indicating the function number and the three parameters to be passed into these routines are respectively held in the D, X and Y registers of the CPU at the time of invoking the interrupt. Where a single parameter is to be returned it is returned in the D register. If the function is successfully invoked, e.g. no addressing errors and a legal function number etc.then the carry flag of the condition code register upon return will be cleared (=0). If an error occurred the carry bit upon return will be set (=1).

As the Polywog provides two teletext screens T1 and T2, the software is provided such that only one screen is "current" at any given time. The current screen is the one to which and from which all character writing and reading take place. The current screen is selectable by a system function (#46). On each of the current screens an independent cursor position is kept, meaning that writes to each screen can be done in different places without alterating the cursor position on the other line. All references to the cursor described below are as a 16 bit number sub-divided into two 8 bit fields. The most significant 8 bit field is the line number on the screen and the least significant 8 bit field is the character position on that line. Line numbers and character numbers start at 0. Cursor positions specified are with reference to the beginning of the unprotected area of the current screen (Lines 1 to 23). If a cursor address is referred to which places it off the unprotected area of the screen, then in general no action is taken and an error condition_is returned.

3.1.1. Memory Map

This is still to be defined and fixed

3.1.2. Interrupt ~~Driven~~ Software Functions

The interrupt functions are:

0. Check Status of Keyboard ✓
   This function checks the keyboard status and returns information as to whether a character is waiting or not.
   Input Parameters - None
   Parameters Returned - The value of the ~~B~~ A register is 0 if no character is waiting and is $01 if a character is waiting to be read from the keyboard. ✓ *

1. Input One Character from the Keyboard ✓
   This function returns the ASCII value of the key if it came from the ASCII keyboard or the special function value of the key if it came from the function keyboard. Invoking this routine clears the keyboard status flag. This routine waits for a key to be pressed before returning. This routine also converts three of the ASCII codes to equivalent teletext codes. These are:
   $23 (35) (ASCII # to $5F (95) (Teletext #)
   $5F (95) (ASCII underline) to $60 (96) (Teletext long hyphen)
   $60 (96) (ASCII back quote) to $23 (35) (Teletext Sterling Pound)
   Input Parameters - None.
   Parameters Returned - The ~~B~~ register contains the value of the key in the A register ✓ *

2. Input One Line from the Keyboard ✗ Use FLEX
   This function causes a line to be input from the keyboard into the line buffer located from the $C080 (49280) to $C0FF (49407) and sets various pointers to the buffer. The function is terminated by the entry of carriage return or whatever the current terminator character is set to. Whilst in this function the cursor is displayed at its current position on the screen and all the character insert delete and editing facilities active. The conversions as specified for interrupt #1 are also done for this function.
   Input Parameters - None
   Parameters Returned - In Memory address $CC14 (52244) - $CC15 (52245). Line buffer pointer pointing to the first character in the line buffer to be processed.

✓ 3. ~~Reserved for Return Next Character in the Line Buffer~~
   Pause if pause flag set.

✗ 4. ~~Reserved for Find Next Valid Entry~~ Set carriage return delay D

✓ 5. Print Character to Current Teletext Screen
   This function prints the ASCII character provided on the current teletext screen and updates the cursor position.
   Input Parameters - Parameter 1 contains the ASCII value of the character to be printed. Only the

~~least~~ most significant 8 bits (the ~~B~~ A register) are used. ✓
Parameters Returned - None.

. Print ASCII String to Current Teletext Screen
✓ This function prints a string of characters to the unprotected area of the current teletext screen pointed to by Parameter 1 which is terminated with a NULL. The position of the cursor is updated by this function to the next available location on the screen.

Input Parameters - Parameter ~~1~~ 2 is a pointer to the first character of the string. ✓

Parameters Returned - None.

7. Reserved for Print Teletext String to Current Teletext Screen
✗

8. Reserved for Read String from Current Teletext Screen
✗

D [illegible]
Priority One
X length
Y positio

9. Write ~~ASCII~~ Teletext String to Specified Line of Current Teletext Screen
✓ This function enables the user to write a string of characters to any line on the teletext screen whether protected or not. Its principal use is for writing to the status line etc.

Input Parameters -

Parameter 1 - starting position on screen, i.e. line and character number in the same format as the cursor position descriptor.

Parameter 2 - starting address in user memory of the string to be copied. The string must be terminated with a NULL character. ✓

Parameters Returned - An error condition will be returned if an attempt to write past the physical end of screen is made.

10. Clear Nominated Line on Current Teletext Screen
✓ This function is used to clear a nominated line. Its principal use is for clearing a status line or lines in the unprotected area of the screen.

Input Parameters - Parameter 1 is the line number to be cleared in the standard cursor format, the least significant 8 bits of this number are ignored. B

Parameters Returned - Error if an attempt is made to clear a non-existent line, i.e. if Parameter 1 > 5888 (23x256). A

11. Set Normal or Reverse Video Attribute on Current Teletext Screen
✓ This function is used to determine whether current printing to the current screen is in normal video or whether in reverse video. Separate reverse video flags are kept for each of the teletext screens.

Input Parameters - Parameter 1 - if 0 selects normal video attribute. if non 0 selects reverse video.

Parameters Returned - None. B

2. Set Absolute Cursor Position on Current Teletext Screen

✓ This function is used to set the cursor to a specified absolute position on the current screen.

Input Parameters - Parameter 1 is the cursor position in normal cursor format. D

Parameters Returned - Error if an attempt was made to move the cursor off the protected screen. In this case, the cursor is not moved.

3. Reserved for Set Relative Cursor Position on Current Teletext Screen D

x

14. Read Current Cursor Position on current Teletext Screen

✓ This function returns the current position of the cursor on the current screen relative to the beginning of the unprotected screen area.

Input Parameters - None. D

Parameters Returned - The current cursor position in standard format.

15. Read Character at Current Cursor Position on Current Teletext Screen

✓ This function is used to read the stored byte at the current cursor position.

Input Parameters - None. B

Parameters Returned - The value of the character read.

16. Reserved for Set Unprotected Area on Current Teletext Screen X

✓

17. Clear Teletext Screen

✓ This function is used to clear the unprotected area of the current teletext screen and set the cursor to the home (top left) position on the screen. ✓

Input Parameters - none

Parameters Returned - None.

18. Set Screen and Display Characteristics

✓ needs modification This function is used to set teletext and graphics characteristics as follows:

bit 15 - ignored D

bit 14 - mix/priority bit 1= mix

bit 13 - 1 = Display G1 (Graphics 1 screen)

~~00=40~~
~~01=20~~
~~11 or 10=80~~

bit 12 - 1 = Display T1 (Teletext 1) in 80 or 20 character mode. See bit 3 for selection of 20 or 80 characters.

bit 11 - 1 = Display T1 (Teletext 1) in 40 character mode.

00 = 40
10 = 20
11 = 80

bits 10 & 9 - Select colour for T1(80) and T1(20)
00 = blue, 01 = green, 10 = red, 11 = white

bits 8 & 7 - not used

bits 6, 5 & 4 - Select B (Background) screen colour. The colour is as defined for teletext characters. i.e.

1 = Red (Binary 001)
2 = Green (Binary 010)
3 = Yellow (Binary 011)
4 = Blue (Binary 100)
5 = Magenta (Binary 101)

```
          6 = Cyan (Binary 110)
          7 = White (Binary 111)
      Bit 3 - Select T1(80) or T1(20) mode where
          1 = T1(80) and
          0 = T1(20)
      See Bit 12 for displaying in this mode.
      Bit 2 - Select G1(480) or G1(240) mode where
          1 = G1(480) and
          0 = G1(240).
      See Bit 13 for displaying G1.
      Bit 1 - 1 = Display G2.
      Bit 0 - 1 = Display T2.
```

Display T1

Input Parameters - Parameter 1 - a 16 bit integer where each bit is set or reset according to the functions required above.
Parameters Returned - None.

19. ✓ Read Display Mode

needs modifying

This funtion is used to read the current status of the display mode.

Input Parameters - None.
Parameters Returned - 16 bit integer where the bits have the same meanings as in function 18.

D

20. Set Real Time Clock String

✓

This function copies an 8 byte string in ASCII format into the real time clock (which is updated at one second intervals). The first 2 bytes of this string are hours, the next byte a separator (e.g.) the next two bytes minutes, another separator, and the last two bytes seconds. The separator used in the string is as defined by the user.

Input Parameters - Parameter 1 - the address of the string in memory to be copied into the real time clock.
Parameters Returned - None.

D

21. ✓ Read Real Time Clock String

This function copies real time clock string to a string in user memory specified by the user.

Input Parameters - starting location of the desired string in user memory.
Parameters Returned - None.

D

22. Set Real Time Clock Attributes

X

This function is used to start and stop the real time clock and to enable it to be displayed on the screen at one second intervals.

Input Parameters - Parameter 1 - if this is 0 the clock neither runs nor displays. If this parameter is positive the clock is run and may be set and read using functions 20, 21, 23 and 24 but is not displayed to the screen. If this parameter is not specified, the clock is run and displayed in the last 8 bytes of line 23.
Parameters Returned - None.

D

23. Set Position of Time Display on Telextext One Screen
X This function allows the user to choose the position on the teletext screen in which the clock is to be displayed. If not set, the clock is displayed in the last 8 bytes of line 24.

Input Parameters - Parameter 1 is the position on the screen in standard cursor format of the first character of the string. X

Parameters Returned - An error is returned if an attempt is made to set the time string within 7 bytes of the end of the physical screen.

24. Return Time in 20 millisecond Increments
X The function returns an integer representing time in 20 millisecond increments. It is an integer in the range ~~0~~ -32767 to 32767 -65535 and wraps around to 0 on reaching the upper limit. X

Input Parameters - None

Parameters Returned - Time integer.

25. Wait
X This function is used to generate delays in 20 millisecond increments (50ths of seconds) . Programme execution is halted for this period of time.

Input Parameters - The desired time delay in 20 millisecond incremenets is placed in Parameter 1, e.g. if Parameter 1 = 50 a time delay of one second will be generated. X

Parameters Returned - None.

26. Set Timer Registers This function allows access to the ✓ registers in the MC6840 programmable timer module in the Polywog. Typical uses would be for timing a signal on the user port or for using the three timers to generate output sgnals for Polyphonic music. Each register is one byte in length and may be accessed individually in any order.

Input Parameters - Parameter 1 is the register number between 0 and 7.

- Parameter 2 is the data to be written into the selected register. B

For both above parameters only the least significant 8 bits are used. X

Parameters Returned - An error if an attempt is made to write to a register numbered greater than 7.

27. Read Time Registers
✓ This function is the complement of function 26 and is used to read the current byte values of the timer registers.

Input Parameters - Parameter 1 is the number of the register to be accessed (0 - 7). B

Parameters Returned - the function assumes the value of the specified register. As this is a byte value bits 8 - 15 will be zeros. An error will be returned if an attempt is made to read a register numbered greater than 7. B

28. Set the Timer Interrupt Vector

✗ This function is used to inform the operating system of the starting address of a user written interrupt service routine which is to be performed when the timer times out as set by relevant timers registers. Any attempt by the user to set this vector disables the real time clock and conversely if the real time clock is re-enabled the user vector is disabled.

Input Parameters - Parameter 1 is the address of the interrupt service routine.

Parameters Returned - An error if an attempt to set the user vector above $BFFF.

29. Reserved for SettingThe User Port Interrupt Vector
✗

30. Reserved for User Interrupt Enable/Disable
✗

31. Set the Break (Escape) Key Interrupt Vector

✓ The break key will always cause an interrupt to user specified address unless a program is being run from the internal machine code monitor program.

DISABLE
ENABLE

Input Parameters - Parameter 1 is the address of the break service routine.

D

Parameter Returned - An error if an attempt to set the user vector above $BFFF.

32. Reserved for Setting the Software Interrupt 2 Vector
✗

33. Reserved for Setting the Software Interrupt 3 Vector
✗

34. Reserved for Setting the User Defined Keys Interrupt
✗ Vector

35. Reserved for Enable/Disable Interrupts of User Keys
✗

36. Read the User Keys

✗ This function is used at any time to determine instantaneously which key is pressed. If two keys are pressed simultaneously the result will be non zero but unpredictable.

Input Parameters - None.

Parameters Returned - The function assumes the value of either zero, for no key pressed, or 1 - 6 depending on the respective key pressed.

✗37. Reserved for Setting the Help Number

38. Reserved for Invoking the Help Function From Within a
✗ Program.

✗39. Move a Data Block in Memory

This function is used to perform a multi-byte move of data in memory. The move is always checked for overlap and is correctly performed regardless of the direction of the move.

Input Parameters - Parameter 1 is the starting address of the block in memory to be moved.

- Parameter 2 is the address of the last byte in the block to be moved.

- Parameter 3 is the address of the first byte

of the destination.
Parameters Returned - An error is returned if an attempt is made to move data in or out of memory above $DFFF. The move is not performed.

40. Set User Port
✓ This function is used to set the user port high or low.
Input Parameters - If parameter 1 is zero, the user port will be set low. If parameter 1 is non zero or not specified, the user port will be set high. B
Parameters Returned - None.

41. Read User Port
✓ This function is used to read the user port.
Input Parameters - None.
Parameters Returned - The function assumes the value of zero if the port is low and assumes a value of one if the user port is high. B

42. Reserved for Reading the Light Pen Register

43. Select Memory Map 1
✓ This function is used to select the standard memory map 1 for the system memory.
Input Parameters - None. ✓
Parameters Returned - None.

44. Select Memory Map 2
✓ This function is used to select the memory map 2 of the dynamic address translator.
Input Parameters - None.
Parameters Returned - None.

45. Write to Memory Map 2
✓ Initially after power-up memory map 2 is initialised to the same as memory map 1. This function may be used to alter memory map 2 according to the user requirements. These bytes are copied into DAT RAM from the specified address. Only bytes 1 - 4 of the bytes specified are used.
Input Parameters - Parameter 1 is the address of the start of the string. ✓
Parameters Returned - None.

46. Select Current Teletext Screen
✓ This function is used to select which teletext screen is current for writing and reading.
Input Parameters - Parameter 1 - if Parameter 1 has the value 2 teletext 2 screen will be selected. If parameter has the value 1 or has any other value teletext one screen will be selected. A
Parameters Returned - None

47. ✓ Reserved for Selection of 24 Line Display on Teletext Screens ✓

48. ✓ Reserved for Selection of 12 Line Display on Teletext Screens ✓

49. Reserved for Selection of Top Page in 12 Line Display Mode ✓ ✓

50. Reserved for Selection of Bottom Page in 12 Line Display Mode ✓ ✓

51. Read Current User's Name ✓
This function causes a string of 12 bytes to be copied into the users memory at the specified location. The string may be chosen to be either the first name or surname of the user currently logged-on.
Input Parameters - Parameter 1 - if 0 selects the first name. If non 0 selects the surname B
Parameter 2 - specifies the starting address of the 12 byte string and user memory to be formed. X
Parameters Returned - Error if specified string address above $DFFF

52. Reserved for Sending Message to Master Disk Unit

53. Reserved for Sending and Receiving a Message X

54. Reserved for Reading the Touch Screen X

55. Invoke Internal Machine Code Monitor Program X
X This function causes a transfer of control from the user program to the machine code monitor. The user program may be resumed by typing "GO" to the monitor.
Input Parameters - None.
Parameters Returned - None

$\bigcirc$56. Log Off
X This function is used to terminate the user program, remove the user's name from the name area, and return the Polywog to the log-on state. Further functions will be added to the log off procedure when communications and security are fully implemented.
Input Parameters - None
Parameters Returned - None.

57. Input Character from Serial Port
X This function is used on the standalone Polywog for reading the serial port attached.
Input parameters - none
Parameters returned - the character from the port.

58. Output Character to the Serial Port
X This function outputs a character to the Serial Port.
Input parameters - P1 = the character to be output.
Parameters returned - none.

$\left.\begin{array}{l}59\\60\\61\end{array}\right\}$ Used by FLEX to get to disk

.1.3. Log On Procedure

After switching the POLYWOG on, if disk is present, FLEX and the BASIC subroutines are loaded. The ROM operating system whether disk is present or not, then enters a user log on routine which

- welcomes the user to POLYWOG
- requests entry of the USER'S FIRST NAME
- requests entry of the USER'S SECOND NAME
- stores these in the protected area
- if disks are not present, enters BASIC, else enters FLEX which calls a COMMAND file which enters BASIC and loads and executes a BASIC program called MENU.

As the user may not know anything about computers, he must be reminded with a flashing message during the requests for his names to "PRESS THE ENTER KEY WHEN HAS FINISHED TYPING HIS NAME".

3.1.4. Keyboard and Function Keys

Auto repeat

Polywog is equipped with a standard ASCII keyboard which has had various keys relabelled to make it a Teletext keyboard. In addition a function keyboard with 16 keys has been added.

The ASCII to Teletext character changes are:

$5B (91) is changed from a left square bracket to a left arrow.

$5C (92) is displayed as a 1/2 rather than a back slash

$5D (93) is displayed as a right arrow rather than a right square bracket

$5E (94) is displayed as an up arrow rather than an inverted v

$7B (123) is displayed as a 1/4 rather than a left curly bracket

$7C (124) is displayed as a pair of vertical lines rather than a single vertical line

$7D (125) is displayed as 3/4 rather than as a right curly bracket

$7E (126) is displayed as divide sign rather than a tilder

Further, the DEL key is used to stop the scrolling when listing. Any key may be pressed to restart it. & also stop BASIC program
with fn 9o symbol off & on
The following code changes are made in the keyboard input routine:

$23 (35) is entered using the # key. This is changed to $5F (95) which displays as # in teletext.

$5F (95) which is the underline in ASCII, is changed to the long hyphen $60 (96) in teletext.

$60 (96) which is the right single quote in ASCII, is changed to the Pound Sterling sign in Teletext.

The various function s keys have differing effects depending on their function but can be broadly classified into three main areas:

1. The ASCII code or function key value (with bit 7 set high to indicate function key) is available as a character to be read from the keyboard.

2. Depression of the key causes an interrupt which is capable of altering the normal program sequence flow.

3. Immediate action is taken on the key and the function is performed followed by a normal return to processing the user program.

Except as noted below the key values are available to any normal character read and do not take immediate action.

The functions of the special keys are:

Value $7F (127)
This causes the listing from the screen to be stopped. It may be restarted by pressing any key. This is entered from the DEL key.

ESCAPE
This key is used as the break key to enable termination of programs where desired or to interrupt out of a program to a user defined location. This key is always active and will perform the equivalent of a non maskable interrupt (NMI) to a user definable location. The 'break vector' may be set by using software interrupt function No 31. At the end of the break servicing routine a return from interrupt or equivalent must be performed in order to free the stack and resume normal program execution. - This key may be trapped in an ON ERROR routine.

CURSOR UP (Upward Arrow)
This key returns the code $80 (128) from the character in routine.

CURSOR DOWN (Downwards Arrow)
This key returns the code $81 (129) from the character in routine.

CURSOR LEFT (Left Arrow)
This key returns the code $82 (130) from the character in routine. If the input line function is being used this causes the cursor to be backed up over a previously entered character# in order to be able to edit that character.

CURSOR RIGHT (Right Arrow)
This key returns the code $83 (131) from the character in routine. If the line input function is being used and the cursor has been backed up over input character, this key cuases the cursor to be stepped to the right but not beyond

the last character entered.

INSERT CHARACTER

This key returns the code $84 (132) from the character in routine. If the line input function is being used this key allows characters to be inserted at the current cursor position.

DELETE CHARACTER

This key returns the code $85 (133) from the character in routine. If the line input function is being used this key causes characters to be deleted from the line input function at the current cursor position.

UNDEFINED

There are two undefined keys at this stage which have the codes $86 (134) and $87 (135) respectively.

HELP

This key returns the code $88 (136) from the character in routine and is reserved for the HELP function.

CALCULATOR

This key returns the code $89 (137) from the character in routine and is reserved for the calculator function.

USER 1 TO USER 6

These keys may be read using interrupt function 34. Later enhancements will allow these keys to cause an interrupt (equivalent to a non-maskable interrupt) to a user specified location.

3.2. BASIC INTERPRETER

3.2.1. Interface PRINT With FLEX

The PRINT statement is used by BASIC to display characters on the screen. As all printing by BASIC is to be in teletext and the screens normal mode is ASCII, all strings to be printed must be preceded by the SI character $1F (31) to switch it into teletext mode, and followed by an SO character $1E (30) to switch it out of teletext mode. The characters are released individually to the screen using interrupt function 5. This routine updates the cursor position by one. The BASIC interpreter must handle all other positioning of the cursor for line feeds and tab column control. Interrupt function 12 is used to set an absolute cursor position. Should an error occur, the error number is returned in register D and the condition carry flag is set. The error number is placed into the BASIC error field and a branch to the error routine made.

.2.2. BASIC Language Alterations

The following additions and changes to BASIC have been made:

1. RND(X) for X > 0
   At present RND(X), where X is an integer such that
   If X = 0 a value between 0 and 1 is returned.
   If X > 1 the previous number is returned again.
   If X < 1 a new random sequence is set up.
   Change this so that
   If X > 1 then an integer between 0 and X is returned.
   RANDOM resets the random sequence r ndomly.

2. PRINT @ (X,Y), string
   This enables the user to print starting at the specified row and column. This requires calling interrupt number nn, with the cursor position placed in registers D and X to position the cursor, followed by a call to interrupt nn to execute and ordinary PRINT. The address of the string is placed in Register D, and its length in register X.

3. USR(I%, P1, P2, P3)
   This changed USR function allows the programmer to
   - Call a machine language subroutine with up to 3 parameters specifying the address as part of the command
   -Call an operating system interrupt routine with up to 3 parameters.

   If a parameter is not specified, USR sets a default value of -1 ($FFFF) and passes this across to the subroutine.
   The USR function carries out the following operations:
   a Stacks the D, X, Y registers
   b Puts the parameters P1, P2, P3 in the D, X and Y registers respectively.
   c If I% is > 255, the subroutine at address I% is executed. Otherwise interrupt I% to the operating system is called.
   d The subroutine, whether interrupt driven or not, may return 1 value in the D register. This is treated as an error code if the carry condition is set. Otherwise, the USR Function takes the value of the D register, unstacks the registers and returns to BASIC.
   e If the carry condition is set, the value in the D register is loaded into the BASIC error code, the registers are unstacked a branch is made to the BASIC error routine.

4. SPC(I%)
   Extend this function so that it can be used in a more general way. At present it can be only used to create I% spaces in a print line. This must be able to create a string I% spaces long. It may be necessary to name it as a string function in SPC$(I%) rather than SPC(I%). It could then be used as A$ = SPC$(100) to create a 100 character string of spaces. If possible, allow the programmer to specify the character with which to fill the string. i.e make the function SPC$(I%,nn) where nn is the decimal value of the ASCII character, or SPC$(I%,"X")

where X is the character.

3.2.3. Fitting BASIC into the 16K ROM

The BASIC interpreter must be made to fit within a 16K ROM. This must be achieved by compressing code and cutting out unnecessary functions and statements. Disk handling functions could be deleted and loaded into RAM on BASIC initialization.

3.3. GENERAL SUBROUTINES

A driver must be supplied to use the port as the printer port so that the PRINT#0 and the P command in FLEX are directed to the printer. The driver may be held on a file and loaded automatically.

3.4. FINE GRAPHICS SUBROUTINES

3.4.1. Specifications

For the fine graphics, it is necessary to set up two work fields to hold

1. Current screen being written to (a 2 bit field where value 0 = G1(240), value 1 = G1(480) and value 2 = G2).
2. Current colour being used (a 2 bit field) where:
   - 0 = Blue
   - 1 = Green
   - 2 = Red
   - 3 = White

Although these fields may be as small as 2 bits each, a byte field would be preferable.

Many of the graphics subroutines use the concept of a boundary being defined inside which the specified operation takes place. The definition of the boundary is held in a string in either of 2 modes:

(a) As a BOUNDARY. This definition is created by a utility from a boundary defined either by a light pen or a digitiser. The form of the string is as follows:

Bytes 1 and 2. The byte offset from the top left corner where each byte represents 6 pixels in a row.
Byte 3.
Bits 2, 1 and 0 contain the value which indicates the first pixel defined in the boundary.
Bit 3 = 1
Bits 7-4 are all zero.
<u>Bytes 2 and 3 are then reversed.</u>

The rest of the string is held in groups of 3 bytes where in each group

Bytes 1 and 2 contain the byte offset from the start byte.
Byte 3
Bits 2, 1 and 0 specify the pixel
bit 3 = 1 indicates a right boundary
and bit 4 = 1 indicates a left boundary
bits 7-5 are not used.

The points are defined in the screen scanning order and all points on all lines are defined.

(b) As a LINE DEFINITION. The points defining the end points of the lines are held in groups of 3 bytes, each group describing a point. Within each group:

Byte 1 holds the pixel row number
Bytes 2 and 3 hold the pixel column as a 2 byte integer.

If Bit 7 in byte 2 has Value 1, this indicates that the "pen is to be lifted" in drawing the string and a new line is to be started. In BASIC this is achieved by adding 32768 to the column number.

Bit 3 =1 Right bounda
Bit 4 =1 Left "

3.4.2. Subroutines

The subroutines for handling the fine graphics are:

~~To set a bit~~ ? $\boxed{\%1 = 256}$
~~Add 256 to rows~~

+compute Z = %1 + φ
Z1% = USR(Z,

1. GSET %1 + φ
 P1 the pixel row
 P2 the pixel column
 P3 not used P3 = φ for reset

GSET displays the defined point, on the current screen in the current colour. If the byte colour is the same as the current colour, the bit is simply switched on. Otherwise, all bits in the byte are switched OFF, the selected bit is put to ON, and the colour is changed to the current colour.

Function

2. GPOINT %1 + 3
 P1 pixel row
 P2 pixel column ~~[illegible]~~
 P3 ~~not used~~

Uses
A% =
m GPOINT, Z0,

GPOINT returns 4 if the point is off and the value indicating the colour if the point is on where 0 = blue, 1 = green, 2 = red and 3 = white.

3. GPRINT %1 +6
 P1 address of string descriptor of the string holding the definition.
 P2 Byte 1: Pixel row.
 Byte 2: Value 1 = row is negative, value 2 = column is negative, value 3 = both row and column are negative.

+4 ⇒ Black

0 1 1 1
φ

P3 pixel column
GPRINT prints the graphics as described in the string. P1 must be specified.

(i) If the string contains a graphics SCREEN DUMP, then this is simply dumped out at the point specified. The SCREEN DUMP must be of the form:
    Bytes 1 and 2: indicate the screen byte address from which it was dumped and
    Byte 3 has bit 4 = 1.
  Bytes 2 and 3 are then reversed
  The dump follows these first 3 bytes.
  If P2 and P3 are not given, then the value stored in the header is used ie the picture is printed at the same place from which it was saved.

(ii) If the string contains a BOUNDARY, then the boundary is displayed in the current colour, the start position being altered to the position given in P2 and P3. The setting of pixels is handled as in GPOINT. If P2 and P3 are not given the boundary is displayed in the position defined. The boundary is stored in the boundary work area.

(iii)
  If the string contains a LINE DEFINITION, then the lines are joined in the current colour. The setting of pixels is handled as in GPOINT. If P2 and P3 are specified, then this value is taken as the coordinates of the 1st point, and all the following points are altered by the difference. By defining 2 points in a string by this method, a line may be drawn simply.

GPRINT only stores a boundary in the boundary work area if the string is of type BOUNDARY.
If P1 is zero, then if there is a valid boundary in the boundary work area, then this is displayed in the current colour.

4. GMOVE   21 ~~24~~ + 9   ~~[illegible]~~
  P1 address of string descriptor of string containing boundary.
  P2 address of string descriptor of string containing: ~~[illegible]~~
      byte 1 current row
      byte 2 has value 1 = row negative, value 2 = column negative, value 3 = both row and column negative.
  —1 bytes 3 and 4 contain the current column. ~~[illegible]~~   FFFF
      byte 5 contains the new row.
      byte 6 has value 1 = row negative, value 2 = column negative, value 3 = both row and column negative.
      bytes 7 and 8 contain the new column.
  Note that if some of the current boundary is outside the screen, then those areas outside the screen are not moved.
  Example:
      In BASIC this string can be set up simply as follows:
      for current row 40, column 200, new row 10 column 30 with the boundary description in B$:
      Set up A$ = CHR$(40) + CHR$(0) + CVTZ$(200) + CHR$(10) + CHR$(0) + CVTZ$(30)
      and the subroutine call as

A% = USR(nnnnn,PTR(A$),PTR(B$))
where the GMOVE subroutine is stored at address nnnnn.
This moves the area defined with the current boundary to the new area and clears all unaffected positions within the current boundary.

If the string contains a LINE DEFINITION, this is converted to a BOUNDARY and stored in the boundary work area before the GMOVE is actioned.
If P1 is specified, GMOVE always stores a boundary in the boundary work area.
If P1 has value zero, then if there is a valid boundary in the boundary work area, then this is used.

21

5. GFILL ~~[illegible]~~ +12

P1 Address of string descriptor containing the boundary.
P2 Byte 1 pixel row,
Byte 2 pixel pattern + row/column negative indicators. The pixel pattern is held in bits 7 to 2 and the negative indicators in bits 1 and 0.
The pixel pattern uses bits 7 to 2 to indicate which bits are ON in the pattern where
1 = ON, and
0 = OFF
P3 Byte 3 Pixel column

The GFILL subroutine fills, within the specified boundary in the current colour, all bytes in the pattern given. If no parameters are specified, except the pixel pattern, the boundary is the whole screen. If no parameters are given, the whole screen is cleared to black.
If the string contains a LINE DEFINITION, this is first converted into a BOUNDARY and stored in the boundary storage area before actioning of the GFILL.
If P1 is specified, GFILL always stores a boundary in the boundary work area.
If P1 has value zero, then if there is a valid boundary description in the boundary work area, then this is used.

71

6. GMERGE ~~[illegible]~~ +15

P1 address of string descriptor of string containing boundary.
P2 byte 1 = pixel row, byte 2 has value 1 = negative row, value 2 = negative column and value 3 = both row and column are negative.
P3 pixel column.

GMERGE merges the other graphics screen onto the current screen within the boundary specified. The current screen has the highest priority such that when 2 bytes in the same position on both screens are of different colours, then the byte from the current screen is used. If 2 bytes have the same colour, then the pixels are ORed.
If no parameters are specified, the whole screens are merged.
If the string contains a LINE DEFINITION, this is converted to a BOUNDARY and stored in the boundary work area before the actioning of GMERGE.
If P1 is specified, GMERGE always stores a boundary definition in the boundary work area.
If P1 has the value zero, if there is a valid boundary

in the boundary work area, then this is used.

7. GSWAP %/ +18

P1 address of string descriptor of string containing the boundary.

P2 pixel row

P3 pixel column

GSWAP exchanges the areas within the defined boundary between the two graphics screens. If no parameters are given then the whole screens are swapped.

If the string contains a LINE DEFINITION, this is converted to a BOUNDARY and stored in the boundary work area prior to actioning GSWAP.

If P1 is specified, GSWAP always stores a boundary in the boundary work area.

If P1 is zero, and if there is a valid boundary description in the boundary work area, then this is used.

8. GRAPHICS %/ +21

~~0 = no graphics~~
1 = G1(240) only
2 = G1 + VS
3 = G1 + G2(240)
4 = G1 + G2(240) +VS
5 = G1(480)
6 = G1(480) +VS

P1

Value 1 = G1(240) required

Value 2 = G2 required

Value 3 = G1(240) and G2 or G1(480) required.

This function must be used at the start of the program. On starting a program, all graphics areas are available as program area. This function saves a screen work area (8K) and a boundary work area (8K) as well as the required graphics screen areas (8K each) for the graphics.

65K

9. GSAVE %1 + 24

3.5. UTILITY PROGRAMS

1. Burroughs down line load program to load files from the Burroughs B2860

2. POLYWOG program to get data loaded down and load it onto a disk file.

3. POLYWOG digitizer program

This must allow both pictures and boundaries to be loaded and stored on disk for later inclusion in programs. These should be stored in either the SCREEN DUMP format or the BOUNDARY format, each with an extra 2 byte header indicating the length of the stored string. Editing facilities to read strings back in must be included.

4. IMPLEMENTATION - PRIORITY 2

Priority 2 implementation is solely concerned in implementing the POLYWOG communications.

4.1. COMMUNICATIONS

4.1.1. CONFIGURATION

The communication system is required to transfer disk and print data between the Master and the Polywogs.

The system requires that:

a. The Communication rate should be 19.2K baud synchronous, to minimise data transfer times.

b. Up to at least 16 Polywogs can be connected to the Master by daisy-chaining. Each Polywog has a cable with a plug to enable connection to the chain, as well as a socket to allow further Polywogs to be connected.

c. All Polywogs should be physically identical. The user should not have to set addresses or perform any other function to enable the Polywogs to be distinguished by the Master.

d. A Polywog that has been powered off should still permit communications to "downstream" Polywogs.

e. Disconnection of any Polywog should not affect operation of any Polywog that is still connected to the Master.

f. Connection of an additional Polywog to the chain at any time should not affect the operation of any already connected, and the new Polywog should be immediately available for on-line use.

g. Error detection and recovery should be sufficient to prevent system hangups, loss of messages, or spurious messages due to noise on the line or problems such as bad connections.

The communications hardware consists of a Motorola 6854 Advanced Data Link Controller chip in each Polywog as well as in the Master. This chip operates in the synchronous HDLC/SDLC format. An external clock is required, and is supplied by the Master. The basic message format handled by the chip is as follows:

Opening Flag	The bit pattern 01111110 identifies the start of the message. This is generated by the chip.
Address	Eight bits identifying the device addressed. These eight bits are supplied by the 6809 software.
Control	Eight bits identifying control functions. Also

supplied by the 6809 software.

Information
: Any number of bits (zero upwards) of data. Supplied by 6809 software.

Frame Control Sequence (FCS)
: 16 bits of cyclic redundancy check information generated by the chip when the "end of message" signal is given by the software.

Closing Flag
: The bit pattern 01111110 is generated by the chip to indicate the end of the message. This flag can also be the starting flag for the next message.

To achieve data transparency, the technique of "bit-stuffing" is applied to the address, control, information and FCS fields. After every five 1's, the chip inserts a 0. The receiving chip removes the 0 after five 1's. If a sixth 1 is received, the receiver knows that either a flag or an "abort" (a continuous series of 1's) has been transmitted.

While not transmitting a message, the chip can be set to either transmit continuous 1's (mark idle) or a series of flags (flag idle).

4.1.2. Loop Mode Operation

The Polywogs operate in "loop mode". Messages transmitted by the master are passed on by each Polywog to the next in the chain. The last Polywog returns its transmission to the Master by a return link that passes through all Polywogs in the chain. The last Polywog is configured in this "loop-back" mode by the "loop set-up" procedure (see below). This procedure also assigns unique addresses to each Polywog.

Polywogs are polled by the Master. A poll string is a short frame (message) with no information field. This poll string is passed down the chain with each Polywog introducing a one-bit delay. The Master eventually receives back its poll string delayed by n bits, where n is the number of Polywogs.

A Polywog wishing to send a message waits until the closing flag of a poll frame addressed to it. It then begins transmission of its own message, which is passed on by downstream Polywogs (each inserting a one-bit delay) and is eventually received by the Master. The diagram below shows the bit string received by the addressed Polywog and the bit string transmitted by it:

```
                 ---------------Poll String------------------------------       -------
                                                                   Master mark idles
          Flag   Address  Control       FCS            Flag      Flag      Address
...1111101111110AAAAAAAACCCCCCCCFFFFFFFFFFFFFFFF011111100111111111111111
...11111101111110AAAAAAAACCCCCCCCFFFFFFFFFFFFFFFF0111111001111110AAAAAAA

-------Message from Polywog--------------------------------------------
Control        Information             FCS              Flag       Mark Idle
```

```
11111111111111111111.....111111111111111111111111111111111111111111111111
CCCCCCCCIIIIIIIIIIII.....IIIIFFFFFFFFFFFFFFFF011111100111111111111111111
```

The "go-ahead" signal for the Polywog is 0 followed by seven 1's. The Polywog changes the seventh 1 to a 0 to form the opening flag for its message. This function is performed by the ADLC chip when it is placed into "go ahead on poll" mode.

This technique can be used to implement group polling. At the end of the message sent by the Polywog is another go-ahead sequence (01111111). Downstream Polywogs can therefore append further messages. The Master will receive its original poll string followed by a series of messages from Polywogs that decided to transmit.

If the Master wishes to send a message to a Polywog, it sends it in place of a poll string. The address of the Polywog is placed in the address field of the frame. The addressed Polywog appends an acknowledgement to the message in the manner described above. The master does not initiate any further transmissions until it receives the go-ahead signal itself.

4.1.2.1. <u>Protocol</u>

The protocol for the Polywog communications is a subset of the IBM Synchronous Data Link Control (SDLC) protocol. The three basic transactions supported are:

Message from Polywog to Master

a. Message from Master to Polywog

b. Reserved for Broadcast Message from Master to all Polywogs

The address field is always used to identify a Polywog. The control field identifies the type of transmission. The control field values to be used are:

(M-P Master to Polywog, P-M Polywog to Master)

Bit ptn 76543210	Acronym	Description - bit 0 sent first, 7 last.
00010111	SIM	(M-P) Set initialisation mode, i.e. initiate loop set-up.
"	RIM	(P-M) Request initialisation mode, i.e. request loop set-up.
01110011	UA	Unnumbered acknowledge - used in loop setup.
10111111	XID	Exchange ID - used in loop setup.
00010011	UI	Unnumbered information - used in loop setup.
00100011	UP	Unnumbered poll. Used for polling Polywogs.
rrr10001	RR	Ready to receive information frame numbered rrr.

rrr1sss0	I	Information frame number sss. Also indicates ready to receive frame numbered rrr.

The numbers rrr and sss are modulo 8, i.e. after 7 they become 0 when incremented.

In the following protocol descriptions, the notation below is used:

a	Address of Polywog selected or polled.
I(x,y)	Information frame control byte with rrr = x, sss = y.
RR(x)	Ready-to-receive control byte with rrr = x.
UP,UI, etc	Control bytes as defined above.
a,I(x,y),data	Frame with address field = a, control byte = I(x,y), information field = data.
m	Number of frame sent by Master.
p	Number of frame sent by Polywog.
m'	Number of next frame expected by Polywog.
p'	Number of next frame expected by Master.

4.1.2.2. <u>Message from Polywog to Master</u>

<u>Message sent by master</u>	<u>Response appended by Polywog</u>
If Polywog not ready to send	
a,UP	no response
If Polywog ready to send	
a,UP	a,I(m',p),data
a,RR(p+1) (processes data, sets p'=p+1)	no response (adds 1 to p)
If errors on line	
a,UP	a,I(m',p),data
no RR message if errors	
......	
a,UP	a,I(m',p),data
a,RR(p+1) (processes data, p'=p+1)	no response (but RR not rec'd)
.....	
a,UP	a,I(m',p),data
a,RR(p+1) (ignores data since expecting frame p+1)	no response (adds 1 to p)

4.1.2.3. <u>Message from Master to Polywog</u>

<u>Message sent by master</u>	<u>Response appended by Polywog</u>
If no errors on line	
a,I(p',m),data	a,RR(m+1) (processes data, sets m'=m+1)
(adds 1 to m)	
If errors on line	
a,I(p',m),data	no response (frame corrupted)
...... a,I(p',m),data	a,RR(m+1) (processes data, sets m'=m+1)
(RR not received)	
..... a,I(p',m),data	a,RR(m+1) (ignores data since expecting frame m+1)

4.1.2.4. <u>Broadcast Message to all Polywogs</u>

This will be implemented in a later version.

4.1.3. <u>LOOP SET-UP</u>

Since all Polywogs are physically identical, a special procedure must be followed on startup to assign a unique address to each Polywog and to configure the loop correctly. This procedure is also executed under the following conditions:

a. If after three attempts, the Master does not correctly receive a message it has transmitted (implying the loop has been broken).

b. On request from a Polywog.

c. If after three retransmissions, the Master does not obtain a RR response to a message addressed to a particular Polywog (implying that that Polywog has been switched off or disconnected)

Each Polywog has a logic-gate switch that can be switched under program control to "loop back" (connect upstream return to own transmitter) or "extend" (connect upstream return to downstream return). When loop set-up is complete, all Polywogs will be switched to "extend", except for the last in the chain, which will be set to "loop-back". In addition addresses will be assigned 01 to the first Polywog, 02 to the second, and so on.

4.1.3.1. Procedure

Message sent by Master		Action taken and Response
0,SIM	All Polywogs:	Switch to Loopback Set m' p and b to 0 Set ADLC to non-loop mode Set ADLC to transmit flag idle Set address to 0
	Response: Flag idle	
1,XID	1st Polywog:	Set address to 01 Respond 1,UI,status,user no where status = 0 - not logged on 1 - logged on as standalone 2 - logged on as online
Check user number		
1,UI,logon status	1st Polywog:	Alter log-on status as requested Switch to "extend" Place ADLC in loop mode
2,XID	2nd Polywog:	Set address to 02 Respond 2,UI,status,user no
Check user number		
2,UI,logon status	2nd Polywog:	Alter log-on status Switch to "extend" Place ADLC in loop mode
3,XID etc		

if there are n Polywogs, then (n+1)th will not respond to XID message:

(n+1),XID	No response
Times out	
n,UI,"loop back"	nth Polywog: Switch to "loop back"
Begin normal polling	

4.1.4. HARDWARE CONFIGURATION

Figure 1 shows the hardware design of the communications interface which is common to both the Master and the Polywog. The 6840 Programmable Timer Module (PTM) is used to generate the 19.2K baud clock in the Master and is used for generating "beeps" and music in the Polywog. These options are selected by strapping. The PTM registers are accessible to the 6809 and are addressable via the RS0, RS1, and RS2 lines. A one-shot generates a "clock present" signal which can be used to interrupt the processor if desired.

The RTS- signal from the ADLC chip (6854) is used to operate the loop-back/extend logic formed from four NAND gates.

A relay activated directly from the power supply switches the up-loop lines directly to the down-loop lines when the power is switched off. This enables the loop to continue operating while Polywogs are powered off.

4.1.5. MESSAGE TYPES

The function of a message is identified by the first byte of the information field. A message is sent from a Polywog to the Master by the following SWI sequence:

```
LDX     buffer start adr
LDY     buffer length in bytes
LDB     message type
SWI
FCB     nn
BCS     transmit error
```

This will transmit the buffer contents to the Master, preceded by the message type byte.

In a later enhancement, the ROM code will also insert a byte indicating the mode in which the Polywog is running. The possible modes will be:

<u>Executive</u>. For running courseware or other programs for which the student has no access to the code.

<u>Teacher</u>. For operation when the key switch is turned on.

<u>Student</u>. When access is provided to the monitor or BASIC for programming.

To receive a response, the above sequence is followed by

```
LDX     recv buffer start adr
SWI
FCB     nn
BCS     receive error
STY     message length in bytes
STB     message type
```

The received response will be placed in the buffer provided by the user.

The high-order bit of B is set if further messages are expected from the Master. The user must execute the receive sequence again to receive these further response messages.

4.1.5.1. Log On

The startup routine in the system ROM stores the Christian and Surnames. Later implementations will transmit this to the Master and check it against a security file.

4.1.5.2. Log Off

This message may be sent by the user program to log off. Control is not returned to the user program. Instead the system displays the log-on request. Later implementations will remove the name and number from the Master.

4.1.5.3. Bootstrap FLEX

This message is sent automatically on startup, and may also be sent by the user to load a new copy of FLEX.

Format:

No data required.

Action:

The Master sends the binary code for the Polywog FLEX. This is loaded into the Polywog RAM by the loader in the ROM.

Response:

FLEX code, in standard binary code format, i.e. address (2 bytes), number of bytes (2 bytes), data to be loaded. An execution address is sent at the end. The load data may be split over several frames to allow for buffer size limits in the Polywog.

4.1.5.4. Data for Printer

Any data to be printed is sent as a message to the Master. The Master prints the data if a printer is available. A later enhancement will allow spooling.

Format:

A string of ASCII characters, with carriage-returns separating lines.

Action:

The Master prints the data out.

Response:

Error code (1 byte) indicating that the printer is not available.

4.1.5.5. Close Printer

When a particular printed report has been finished, a "close printer" message must be sent to release the printer for other users. The "close printer" function is performed automatically at log-off.

Format:
No data required.

Action:
The Master releases the printer.

Response:
None.

4.1.5.6. Help

The HELP function will be released in a later version.

4.1.5.7. Execute FLEX command

Any commands that cannot readily be handled by FLEX in the Polywog (such as copy files) are sent directly to the Master which attempts to process them. Such commands are those that access disk files frequently but do not attempt to access the user memory or the keyboard and do not produce large volumes of data to be displayed on the screen.

Format:
Text of command line, in ASCII.

Action:
The FLEX in the Master is called as a subroutine, using the text string supplied as a parameter. Any console output produced by the Master FLEX is trapped and sent as responses to the Polywog for display.

Response:
Text to be displayed, in ASCII. Several frames may be sent.

4.1.5.8. FMS Call

The FMS (File Management Services) portion of FLEX in the Polywog is replaced by code that sends messages to the Master to perform disk functions. The Master receives these messages and calls its own FMS to process them. Details of the functions are described in the next section. Access to files is controlled by the mode byte included in the message by the ROM software.

Format:
A File Control Block (FCB) in FLEX format, with or without the data portion. The FCB contains all parameters required by the FMS to perform the function requested.

Action:
A corresponding FCB is set up or accessed in the Master. A

FMS call is made using the address of this FCB as parameter. The updated FCB that results (with certain fields erased) is returned to the Polywog.

Response:

The updated FCB, with or without the data portion. This is used to update the Polywog FCB.

4.1.5.9. <u>Further Extensions</u>

Further message types will be included as the need arises. These include the following:

a. <u>Send message to Polywog</u>.

b. <u>Message for Status Line</u> (Master to Polywog).

c. <u>Accept Message</u>.

d. <u>Fetch Memory Data from Polywog</u>.

e. <u>Request Memory Data</u> (Master to Polywog).

4.1.6. <u>DISK OPERATIONS</u>

Disk operations are performed by calling the FMS portion of FLEX with the address of a File Control Block in the X register. The FCB contains all parameters required for the function as well as details of the file being accessed. With the separated disk environment, the same procedure will be followed for disk operations, except that:

a. Drive numbers will not be available to the Polywog.

b. The file name will be only 6 characters since the last two characters are reserved for the insertion of a user code by the Master.

c. Security will be implemented in later versions.

d. Track/sector information is not available to the Polywog.

e. Directory information is not available to the Polywog. However a future extension may be required that information on ones own files be available on a read only basis.

f. A file number will be assigned by the Master when the file is opened. This number will reference the FCB copy in the Master. It must be retained by the user in the FCB and supplied on each disk access. If it is altered by the user, errors may occur. The Master will check the file number on every access to ensure that the user accesses his own files only.

g. Functions involving direct access or update of sectors or directory information will not be available to the Polywog.

4.1.6.1. FCB Format

Byte no	Function
0	Function code. Set by the user to determine the function to be performed.
1	Error Status Byte. Set by FLEX to indicate error conditions.
2	Activity Status. 1=open for read, 2=open for write.
3	Not used.
4-9	File name (6 characters). Set by the user before an Open File.
10,11	Not used.
12-14	File name extension. Set by the user before an Open File.
15	File attributes.
16	File number assigned by Master.
17-20	Not used.
21,22	File size.
23	Random/serial file indicator. Set by user on create, else set by FLEX.
24	Not used.
25-27	File creation date.
28,29	FCB list pointer. Used by FLEX for linking open FCB's into a queue.
30,31	Not used.
32,33	Current record number in file.
34	Byte index. Used for byte read/write.
35	Random byte index for random byte read.
36-46	Temporary storage for file name.
47-52	Not used.
53-63	New name and extension for rename.
59	Space compression flag.
64-319	Sector data buffer.

4.1.6.2. Allowed Function Codes

Code	Function
0	Read or write next byte into sector buffer. An automatic read/write is performed when the buffer is empty/full.
1	Open file for read.
2	Create and open file for write.
3	Open file for update.
4	Close file.
5	Rewind file, i.e. reposition at start.
12	Delete file.
13	Rename file.
15	Get next sequential sector of file.
17	Get random byte from sector.
18	Put random byte into sector.
21	Position to record n of random file.
22	Back up one record in random file.

4.1.7. STARTUP ROUTINE

On power-on, the following actions take place for a student's Polywog:

1. The log-on display is generated and the user names are accepted.

2. A "log-on" message is sent to the Master.

3. The FLEX code is loaded into the user Polywog.

4. The BASIC program on a special startup file is loaded (using FLEX) and control is passed to the BASIC interpreter to execute this program.

5. The BASIC program displays a menu.

6. After a selection is made, the BASIC program loads the appropriate courseware program and executes it.

7. If "Programming" is selected, the BASIC returns control to the system ROM. Logging off is accomplished by the software interupt #56 to the ROM. Commands are available to enable log-off to be performed from BASIC, FLEX or the monitor.

5. IMPLEMENTATION - PRIORITY THREE

The facilities described in this section are not essential for the evaluation but are essential if wider marketing of Polywog is contemplated. Any of these facilities which are included in the evaluation, will be accepted as a bonus.

5.1. BASIC INTERPRETER

The enhancements included extend the BASIC language to handle all the commands handled as subroutines in Priority 1.

5.1.1. TELETEXT COMMANDS

1. CLS
   Clears the current teletext screen to spaces and resets the cursor to (0,0).

2. SCREEN type, colour
   Sets the current teletext screen type to T1(20), T1(40), T1(80) or T2. The colour is specified for T1(20) and T1(80) only. The available colours codes are: 0 = Red, 1 = Green, 2 = Blue and 3 = White. Initially in a program, the screen colour for T1(20) and T1(80) is set to white. No background colour may be set.
   The teletext screen type is set to T1(40) on every RUN, CLEAR and CHAIN command.

3. PRINT@(X,Y),string
   This command first sets the cursor to the specified position on the current teletext screen, and then does a normal PRINT.
   Colour is set on T1(40) and T2 by the inclusion of the colour control characters in the print string.

4. SET(X,Y)
   SET accesses the teletext graphics and displays the specified coarse graphic has 80 by 72 pixels on T1(40) and T2 screens. Graphics may not be displayed on the T1(20) and T1(80) screens. The colour is as currently defined for that character.

5. RESET(X,Y)
   Similiar to SET but it switches the pixel OFF.

6. LINE (X1,Y1),(X2,Y2)
   Draws a line between the two specified points. This SETS the required pixels ON in the colour defined at the points along the specified line.

7. POINT(X,Y)
   Has the value 0 if the teletext pixel on the current screen is OFF, and has the value 1 if it is ON. (This is a function.)

8. BACKGROUND colour
Sets the background colour to any of the 8 colours.

9. DISPLAY screen ON
or DISPLAY screen
Sets the specified screen to DISPLAY.
DISPLAY screen OFF
Switches the displaying of a particular screen OFF.
DISPLAY ALL ON
or DISPLAY ALL
Sets screens T1(40) and T2 to DISPLAY.
DISPLAY ALL OFF
Switches ALL screens OFF.
If any T1 screen is set to DISPLAY, then the other T1 screens are set to OFF. Further, if T1(80) is specified then T2 is also switched to OFF automatically. (Note: When fine graphics commands are implemented in BASIC, this command will be extended to handle the graphics screens as well.)

10. TIME
Returns the current time as an integer variable in 20 millisecond units.(50ths of seconds.) The time is set to 0 when the POLYWOG is switched on. On reaching 65535, it is also reset to 0.

11. WAIT(time 20 millisecond units)
Waits the specified number of milliseconds before returning.

12. SOUND pitch,length
Sounds the speaker as specified. Further commands will be necessary for multiple sound production.

13. EDIT
At present a BASIC program can be editted on exiting from it. The EDIT command must be entered first to preclude accidental editing. The edit mode is switched off on the entering of any RUN, GOTO or GOSUB command.

14. LPRINT
Implement LPRINT as a separate command to handle printing to a line printer as an equivalent to the

```
OPEN "0.PRINT" AS 0
PRINT #0, string
CLOSE 0
```

so that the programmer does not need to worry about the opening and the closing of the file or the awkward PRINT #0 statement.

5.1.2. FINE GRAPHICS COMMANDS

The fine graphics screen handling in BASIC is specified here as suggestions. The X and Y coordinated are all pixel column and pixel row where there are 205 rows on all graphics screens, and either 240 or 480 columns. The suggested BASIC commands are similiar to thse for the teletext screen except that they are all preceeded by a G.

1. GRAPHICS type
   At the start of the execution of a BASIC program, the memory areas used for the Graphics are available to the user for programming. GRAPHICS which must be used as the first statement in the program, is required if any graphics are to be used. There are 3 areas, each of 8K bytes, as well as a smaller area used for boundary definition, used for graphics. Any use of graphics requires use of both the boundary definition area and the 8K screen work area. The screentypes are:
   1 - G1(240) only.
   2 - G2 only.
   3. G1(480) or G1(240) and G2 (Default if the type is omitted.)
   No Graphics commands are available unless GRAPHICS is specified.

2. GSCREEN type
   Sets up the current Graphics screen type. (G1(240), G1(480) or G2).

3. GSET(X,Y)
   Sets the specified point ON in the current colour. If the byte is of a different colour, then all other pixels in the byte are reset, the colour is changed and the pixel is set.

4. GRESET(X,Y)
   Sets the specified point to black.

5. GPOINT(X,Y)
   Returns 0 if the point is OFF otherwise the number indicating the colour is returned.

6. GCOLOUR(colour)
   Sets the colour on the current graphics screen. (0 = Red, 1 = Green, 2 = blue, 3 = white)

7. GPRINT@(X,Y),string
   or
   GPRINT@(X,Y);"filename"
   Displays the graphics defined in the string specified, at the given point. If the filename option is used, the graphics description is read from disk and then displayed. It is not stored in a string.

8. GFILL string variable, pixel pattern
   The string variable must be set up previously to contain a boundary definition. This command fills in to the boundary with the current colour. If a null string or no string is specified, the whole screen is filled in the current colour in the specified pixel pattern. If the pixel pattern is omitted, a pixel pattern of 0 is used. i.e. The command GFILL with no parameters, clears the current graphics screen. The command GFILL A$ clears the area defined by the string A$.

9. GMERGE (X,Y), string containing boundary description.
This merges the other graphics screen onto the current graphics screen within the boundary specified.

10. GSWAP (X,Y), string containing a boundary definition.
Swaps the graphics screens within the specified boundary.

11. CIRCLE (X,Y),radius in pixels
Draws a circle with centre at (X,Y).

12. GSIZE(X,Y),string containing boundary description, enlargement coefficient
Enlarges the picture within the boundary specified, around point (X,Y) with the enlargement specified.

13. GROTATE(X,Y),string containing boundary description, degrees
Rotates the picture within the boundary specified, around point (X,Y) the specified number of degrees.

14. GMOVE(X1,Y1) to (X2,Y2),string containing boundary description
Moves the picture within the boundary defined at (X1,Y1) to (X2,Y2).

15. TOUCH(X,Y)
This monitors the touch screen and returns value 1 if the specified spot is touched, and 0 otherwise. The values of X and Y must be in the range 0 to 15. (This is a function and NOT a ststement).

5.2. OPERATING SYSTEM

5.2.1. FUNCTION KEYS

Insert here the implementaion of those not specified in Priority 1

5.2.2. SECURITY AND LOG ON

Insert full specifications

5.3. SUBROUTINES

1. LIGHTPEN:Returns screen position of light pen on current graphics screen.
   Input: none
   Output: Screen position, a high value if not found.

2. TOUCH: This monitors the touch screen and returns value 1 if the specified spot is touched, and 0 otherwise. The values of X and Y must be in the range 0 to 15.

3. ROTATE:

4. SIZE: