



**OP7100  
Graphics Engine  
OP7100GE Rev. 2.62**

## Table of Contents

<b>OP7100 GRAPHICS ENGINE .....</b>	<b>3</b>
PURPOSE .....	3
OVERVIEW .....	3
HIGH-LEVEL FUNCTIONAL DESCRIPTION .....	4
<i>Macros</i> .....	4
<i>Cells and Buttons</i> .....	4
<i>The Virtual Keyboard</i> .....	5
<i>Time/Date entry/readback</i> .....	6
<i>Initialization</i> .....	6
<b>PROTOCOL.....</b>	<b>6</b>
RS-232 PACKET FORMAT .....	6
MISCELLANEOUS INFORMATION .....	8
USE OF FLASH EPROM .....	9
<b>FINITE STATE MACHINE .....</b>	<b>10</b>
COMMANDS .....	11
<i>Control Commands</i> .....	11
<i>Macro Commands</i> .....	13
<i>Graphics Primitives Commands</i> .....	13
<i>Text Commands</i> .....	15
<i>Miscellaneous Commands</i> .....	15
<i>Scripting Commands</i> .....	18
<i>New (Special) Commands</i> .....	19
<b>APPENDIX I     EXPLANATORY NOTES .....</b>	<b>20</b>
THE WAY MACROS WORK .....	20
INCLUSIVE VS EXCLUSIVE LIMITS .....	20
THE WAY THE SCROLL COMMAND WORKS .....	20
THE WAY THE XOR BRUSH WORKS .....	21
<b>APPENDIX II     RESPONSE FORMATS .....</b>	<b>22</b>
<b>APPENDIX III    MAXIMUM VALUES.....</b>	<b>23</b>
<b>APPENDIX IV     COMMANDS, WHICH CAN BE INCLUDED IN MACRO .....</b>	<b>24</b>
<b>COMMAND INDEX.....</b>	<b>25</b>

## Table of Figures

Figure 1	Command/Response Protocol .....	4
Figure 2	Unsolicited Response Option .....	4
Figure 3	Virtual Keyboard .....	5
Figure 4	RS-232 Packet Format .....	6
Figure 5	x,y Coordinates .....	8
Figure 6	Cell Numbering Scheme (valid in both Landscape and Portrait).....	9
Figure 7	Finite State Machine.....	10
Figure 8	Command handling .....	11

## **OP7100 Graphics Engine**

### ***Purpose***

The primary purpose of the OP7100 Graphics Engine is to allow use of an OP7100 controller from another (not necessarily Z-World) controller via RS232 without requiring Dynamic C for further programming of the OP7100. An OP7100 with a well-documented and versatile serial graphics command interpreter is a product that can be useful to people who don't use other Z-World controllers and to those who do.

### ***Overview***

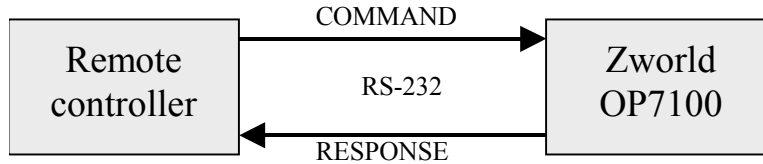
Because of the extra memory on the OP7100, a large portion of memory is used for storing bitmaps, fonts and macros, which can be referenced from the remote controller by index numbers. Three default fonts (small, medium, and large) are provided with the OP7100 Graphics Engine.

Macro and button capabilities give the OP7100 Graphics Engine the ability to run user interfaces autonomously. An OP7100 Graphics Engine interface can run independently of the remote controller.

Button hits are detected and sent back to the remote controller in a manner specifiable by the user. A special pop-up, virtual alphanumeric keypad is predefined to facilitate data entry.

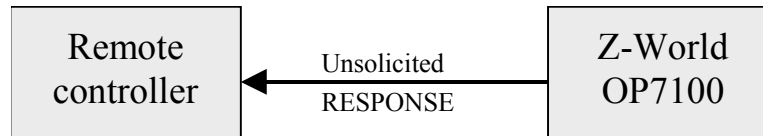
## High-Level Functional Description

A finite state machine (FSM) runs on the OP7100, which will interpret and execute a command protocol for displaying graphics and text primitives.



**Figure 1 Command/Response Protocol**

Graphics commands are sent from a remote controller using the RS-232 serial protocol. Commands will be ACKed and NAKed. Some commands request data to be sent to the remote; this data is sent in the response.



**Figure 2 Unsolicited Response Option**

An unsolicited response can occur with a key-push if the latter has been set up in this manner.

## Macros

The engine has the ability to define and play macros. Macros are collections of drawing or text commands that can be defined remotely and stored without being drawn. Macros can call other macros and be recursive. Macros can be played in “loop-back” mode, where they are played continuously until a StopMacroLoop command is received or played in a macro. Macros may contain time delays. Complex drawing sequences or whole graphics user interfaces (GUIs) can be played as macros and initiated by single commands from a remote host or initiated locally (for example, by another macro)

## Cells and Buttons

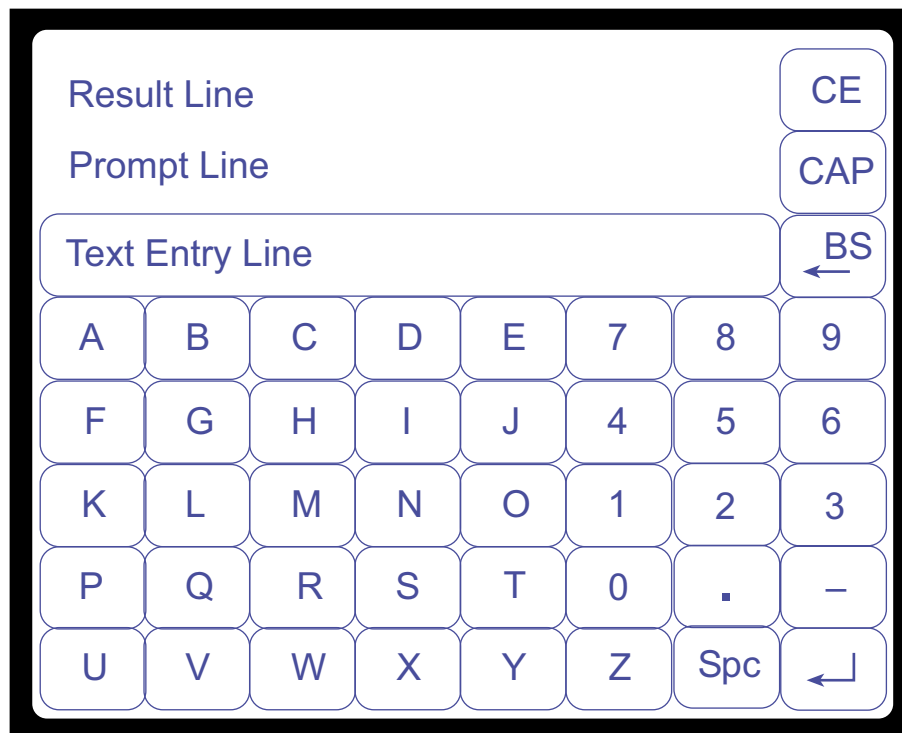
The 240 x 320 pixel, ¼ VGA OP7100 LCD supports both graphics and text. The touch-screen divides that area into an 8 x 8 matrix.

- “Cells” are defined as the cells of the 8 x 8 touch-screen matrix and are predefined by the software and hardware. A Cell, when hit while enabled, sends a one-byte message corresponding to its coordinate on the grid (upper-left corner = 1, next in top row = 2, etc). Cells are disabled by default, and must be enabled by a command from the Remote.
- Buttons are defined as a collection of one or more neighboring cells arranged in a rectangular pattern. A button must be created by a command from the remote controller. A button, when hit while enabled, sends a pre-defined two-byte message, which was defined when the button was created.

Buttons can be linked to cells and macros. The graphics engine will make sure that no two buttons have the same index, and that only the top button is enabled if buttons overlap. Cells may also be linked to macros and set to send codes to the remote, but putting a button over a cell disables the cell (a touch will be interpreted as a button hit).

### ***The Virtual Keyboard***

The virtual keyboard uses  $8 \times 5 + 3$  keys. It contains 26 letter keys, 10 digit keys, plus caps lock (CAP), clear entry (CE), enter, space, minus, decimal point, and backspace keys. A text entry box echoes typed keys. This is a standard way of entering and sending strings and numeric data. In the command to run the virtual keyboard, the user specifies whether to send the data to the host and whether to remove the keypad on hitting the enter key. The last data entered will always be stored locally and can be retrieved by **SendString**, **SendTod**, **SendLongInteger**, **SendFloat** and **SendByte** commands.



**Figure 3** *Virtual Keyboard*

The user can also optionally specify:

- Password mode, where all characters typed are echoed as asterisks
- Data type to be entered (float, long integer, string)
- High and low limits
- Prompt line message
- Text Entry line message

If a wrong type or range of data is entered, the unit will (optionally) beep and display an error message on the Result line.

### Time/Date entry/readback

Time/Date (TOD) entry/readback is in the form of a standard Z-World structure

```

struct    tm    {
    char tm_sec;    // 0 - 59
    char tm_min;    // 0 - 59
    char tm_hour;   // 0 - 23
    char tm_mday;   // 1 - 31 (day of the month)
    char tm_mon;    // 1 - 12
    char tm_year;   // 0 - 150 (1900 - 2050)
    char tm_wday;   // 0 - 6 (day of the week, 0 =
    Sunday)
};

```

The Real-Time Clock resident on the OP7100 is set when TOD is set.

### Initialization

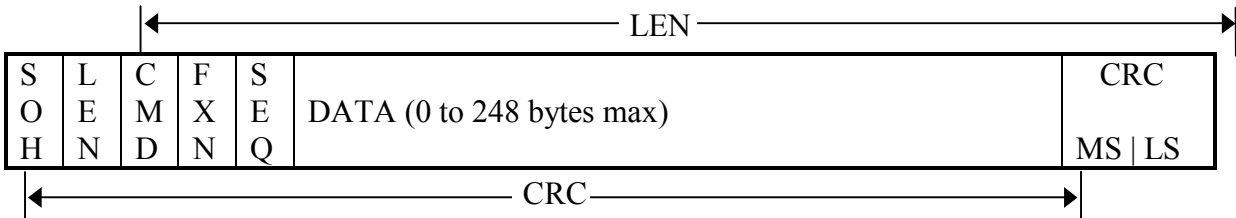
On power up, prior to receiving the 1<sup>st</sup> command from the remote controller, an OP7100 equipped with the Graphics Engine will display the message “Graphics Engine,” and the software version and date. User specified fonts and bitmaps are stored in flash EPROM at the time they are downloaded, and automatically restored.

### Protocol

Commands are transmitted in packets. Most commands can be transmitted in one packet; some commands (such as font and bitmap data) require continuation packets. Commands and data are binary.

### RS-232 Packet Format

Packets are limited to 255 bytes. All packets have the following format:



**Figure 4 RS-232 Packet Format**

The minimum length of the message is 5 bytes. See “Appendix II – Response Formats” for typical command-response messages.

SOH	Start of Header (0x01) This byte flags the beginning of a packet.
LEN	Record Length The length of the rest of the record, including the CRC but excluding the 1 <sup>st</sup> two bytes.
CMD	Command associated with Record. Sent from remote to OP7100GE, echoed by OP7100GE in response.
FXN	Function or Subcommand (used by remote in Command) Used by OP7100GE in Response: ACK (0x06) Command received successfully. NAK (0x15) Flags an error. The error is transmitted in the 1 <sup>st</sup> data byte: 1 = CRC error 2 = Record less than 5 bytes 3 = Command not recognized 4 = Command not implemented as yet 5 = Option not recognized 6 = RTC has failed or is not installed 7 = Command queue full 8 = Command not allowed in a macro 9 = Error adding a macro to the list (probably exceeds maximum length) 10 = END_MACRO command when not building a macro 11 = A command target (add, delete) does not exist 12 = Memory allocation error 13 = Parameter out of range, or object not found 14 = Command other than STOP_MACRO_LOOP if a macro is playing
SEQ	Sequence Number Used by remote to provide additional information. Used by OP7100GE to convey bit-encoded information: b0 set = data has been entered since last message <sup>1</sup> b1 set = cell hit since last message <sup>2</sup> b2 set = button hit since last message <sup>3</sup> b7 set = error occurred since last message <sup>4</sup> This information is kept in global char lastStatus. This is cleared after transmission so that the user will not get the same information over and over again.

---

<sup>1</sup> Retrieve data with SendString (etc) command.

<sup>2</sup> Retrieve cell push with SendLastPush command.

<sup>3</sup> Retrieve button push with SendLastPush command.

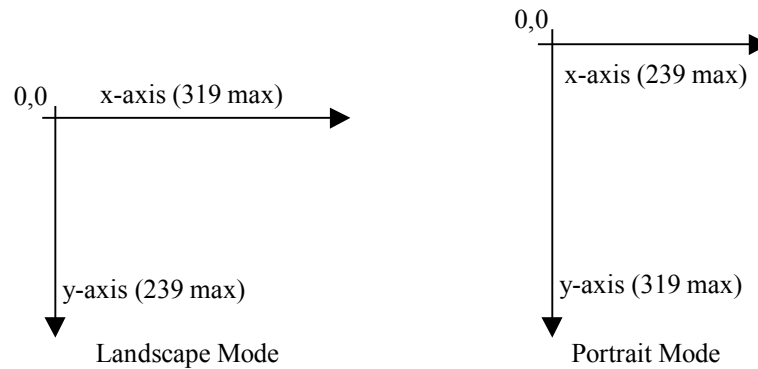
<sup>4</sup> The error is in the 1<sup>st</sup> data byte.

**CRC**            Cyclic Redundancy Check.  
 The 16-bit CRC is at the end of the record and computed for all preceding bytes. The MS CRC comes first at byte [LEN], followed by the LS CRC at byte [LEN+1]. The CRC is the standard Z-World CRC returned by function `getcrc()`.

### **Miscellaneous Information**

- Indexes and identifiers (ID) start with 1 (0 is reserved for “no” or “none”, or ...)
- The sizes of the three standard fonts are:
 

6 x 8	fontId = 1
12 x 16	fontId = 2
17 x 35	fontId = 3
- A coordinate pair takes the form:  
**(int)xx followed by (int)yy,**  
**where xx = x address, yy = y address**



**Figure 5**    **x,y Coordinates**



1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

**Figure 6** Cell Numbering Scheme (valid in both Landscape and Portrait)

### **Use of Flash EPROM**

Fonts and bitmaps sent by the user are stored in Flash EPROM, and restored after RESET. The user must issue the SuperReset command to a new board in order to initialize this feature.

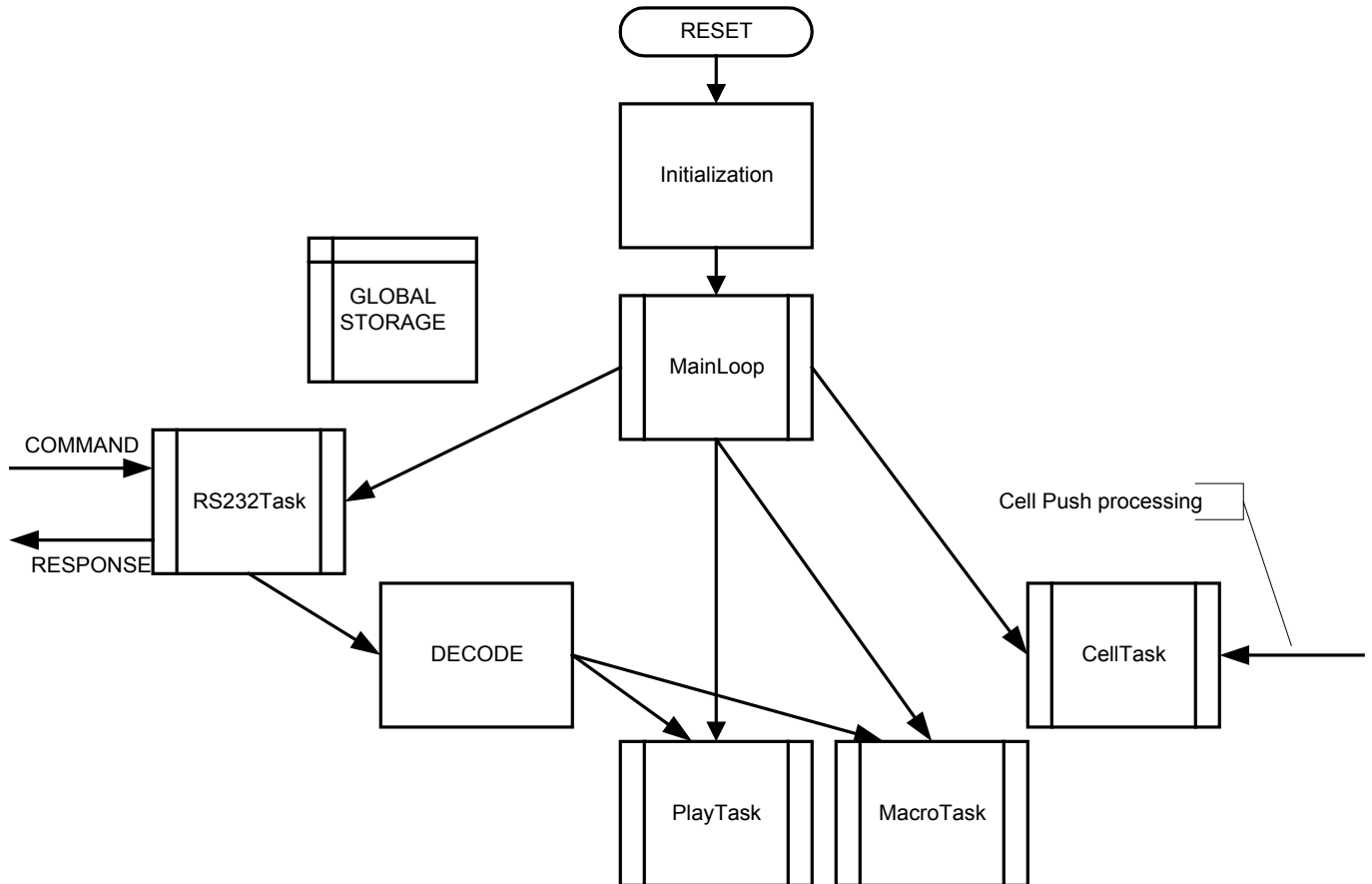
---

*Caution: Use of SuperReset after Fonts and Bitmaps have been stored will clear these items.*

---

## Finite State Machine

The Graphics Engine's implementation is that of a Finite State Machine (FSM). It is a concurrent task application using Dynamic C's costatement feature and functions from the AASC libraries to buffer incoming commands. Graphics primitives from the OP71HW.LIB library are used for rendering. Linked lists are used to store bitmaps, fonts, saved regions, and Macros. Inter-task communication is implemented with queues and global storage.



**Figure 7 Finite State Machine**

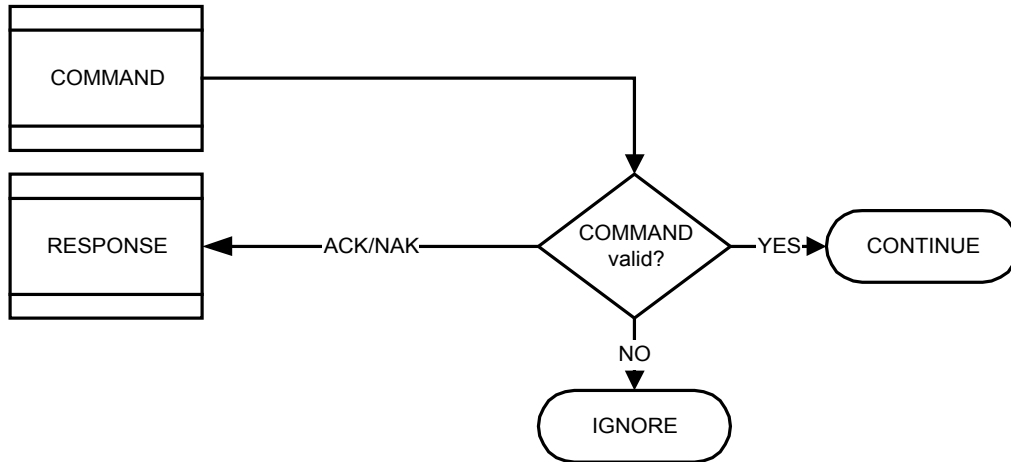
The RS232Task is in charge of receiving the RS-232 packet. It calls the Decode subroutine to partially parse the Command (some of the commands can be executed directly by the task) and passes the information to the PlayTask and/or global storage.

The CellTask is in charge of picking up cell pushes and translating them into button pushes (if necessary).

The PlayTask is in charge of displaying graphics/text command and macros.

The MacroTask is in charge of displaying the macro steps, and to allow the user to queue more than one macro.

## Commands



**Figure 8 Command handling**

A command is always either ACKed or NAKed. A command is acted on if valid (good CRC, valid parameters), ignored if not.

## Control Commands

In the following table omitted fields are not used (don't care). Commands in *italics* are not implemented.

Command	Stream	Note
Init (general)	CMD=10 FXN=0	Reset board, clear screen.  This command is not recommended; the Graphics Engine software performs an Init on reset. See also the SuperInit command.
SendStatus	CMD=12	Response: SEQ = lastStatus: DATA_ENTERED 0x01 CELL_HIT 0x02 BUTTON_HIT 0x04 ERR_OCCURRED 0x80 data[0] = lastError (0 if none)  lastStatus is cleared after transmission.
SendLastPush	CMD=13 FXN=1	Response data <sup>5</sup> = byte index of last cell pushed, 0 = none (see Figure 6).
SendLastPush	CMD=13 FXN=2	Response data <sup>5</sup> = (int) buttonId of last button pushed, 0 = none.

<sup>5</sup> Cells and buttons are "one-shot": a code is returned if the item has been pushed; the next inquiry returns a zero.

<b>Command</b>	<b>Stream</b>	<b>Note</b>
SendString	CMD=14	Send the last string entered with the virtual keyboard or loaded by SetString. ASCIIZ string returned in data (could be a NUL string).
SendLongInteger	CMD=15	Send the last integer entered with the virtual keyboard or loaded by SetLongInteger. Integer returned in data.
SendFloat	CMD=16	Send the last float entered with the virtual keyboard or loaded by SetFloat. Float returned in data.
SendChar	CMD=17	Send the last character entered with the virtual keyboard or loaded by SetChar. Char returned in data. Char is one-shot; the next command returns a 0.
SendTod	CMD=18	Send the date and time of day from the RTC in the form of the 7-byte struct tm (see page 6).
ClearString Buffer	CMD=20	Set string buffer count to zero. Clears Text Entry line if the Virtual Keyboard is active.
DeleteBitMap	CMD=21 FXN=0 SEQ= index	Delete the bit map identified by the index.
LoadBitMap (start)	CMD=21 FXN=1 SEQ= index data = (int)dx,dy	Specifies size of bitmap (must be followed by FXN=2 records). Indexes 251 to 255 are reserved for internal use. See SuperReset command.
LoadBitMap (continue)	CMD=21 FXN=2 SEQ= index data = packed pixel data	Pixel data consists of 1-bit values, packed in bytes in the order of ms-bit on left to ls-bit on right. The number of continuation records is determined by the size of the bit map: size = ((dx * dy) + 7) / 8 bytes.
DeleteFont	CMD=22 FXN=0 SEQ=index (4+)	Delete the font identified by the index. Cannot delete default font (index = 1 to 3).
LoadFont (start)	CMD=22 FXN=1 SEQ=index (4+) data = (byte)width,h eight, startChar, endChar	This cannot be used with the default font indexes (which have the values of 1, 2, 3). Font indexes must be in the range of 4 to 20. A font may not have a startChar of 0 (ASCII NUL is not displayable) See SuperReset command.

<b>Command</b>	<b>Stream</b>	<b>Note</b>
LoadFont (continue)	CMD=22 FXN=2 SEQ= index data = (byte)font data	The number of continuation records is determined by the size of the font: size = $((w+7)/8 * h * (end - start + 1))$ bytes
DisableCells	CMD=24 FXN=0	Ignore all cell hits.
DisableCells	CMD=24 FXN=cell index	Ignore specific cell hit (see Figure 6).
EnableCells	CMD=25 FXN=0	Activate all cell hits.
EnableCells	CMD=25 FXN=cell index	Activate specific cell hit.
SuperReset	CMD=26	Clear all bitmap, macro and font lists except default fonts and Virtual Keyboard.  <b>This command must be used at least once for a board to enable loading of bit maps and fonts.</b>

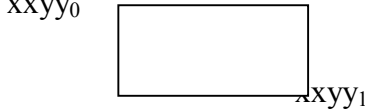
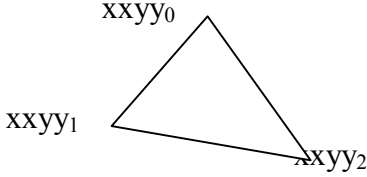
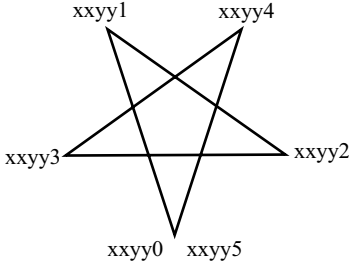
### **Macro Commands**

<b>Command</b>	<b>Stream</b>	<b>Note</b>
StartMacro	CMD=30 FXN,SEQ=ID <sup>6</sup>	This command starts a group of commands, which are included in the macro. Macros are limited to 1000 bytes, but one macro may play another (nested). See “The way Macros work.”
EndMacro	CMD=31 FXN,SEQ=ID	This command ends a group of commands, which are included in the macro.

### **Graphics Primitives Commands**

<b>Command</b>	<b>Stream</b>	<b>Note</b>
ClearScreen	CMD=40	Sets the screen to all 0's (white)
SetBrushType	CMD=41 FXN= 0 CLEAR 1 SET 2 XOR	Set brush to 1 of 3 types. The brush applies to all of the commands following the SetBrushType command. The board is initialized to use the SET brush. See “The way the XOR brush works” for additional information on the XOR brush.
Pixel	CMD=42 data=xyyy	Draws one pixel, using brush.

<sup>6</sup> The byte pair FXN,SEQ form an integer (in little-endian order) defining the Macro ID.

Command	Stream	Note
Line	CMD=43 data= xxyy <sub>0</sub> xxyy <sub>1</sub>	Draws a line, using brush (limits are inclusive).
Circle	CMD=44 FXN= 0 not filled 0xFF filled data= xxyy <sub>0</sub> (int)radius	Draws a circle, using brush.
Rectangle	CMD=45 FXN= 0 not filled 0xFF filled data=xxyy <sub>0</sub> xxyy <sub>1</sub> (inclusive).	The boundaries are parallel to the axes. 
Polygon	CMD=46 FXN= 0 not filled 0xFF filled SEQ= number of sides n data=xxyy <sub>0</sub> xxyy <sub>1</sub> ... xxyy <sub>n-1</sub>	
Polygon (same command, different data.)	CMD=46 FXN= 0 not filled 0xFF filled SEQ= number of sides n data=xxyy <sub>0</sub> xxyy <sub>1</sub> ... xxyy <sub>n-1</sub>	This figure utilizes convex angles and must be closed (the last xxyy is the same as the first xxyy). In this case, n = 6. 
BlankRegion	CMD=47 FXN=0 to clear FXN= 1 to set data=xxyy <sub>0</sub> xxyy <sub>1</sub>	Clears or sets the contents of a rectangular region.
InvertRegion	CMD=48 data=xxyy <sub>0</sub> xxyy <sub>1</sub>	Inverts contents of rectangular region.
StoreRegion	CMD=49 SEQ=index data=xxyy <sub>0</sub> xxyy <sub>1</sub>	Up to 255 rectangular regions indexed by SEQ can be stored and restored.

<b>Command</b>	<b>Stream</b>	<b>Note</b>
RestoreRegion	CMD=50 SEQ=index data=xyy <sub>0</sub> (top left corner)	A stored region does not have to be restored at the place from which it was stored. A new top left-corner coordinate can be specified (-1,-1 = use stored coordinate). The size of the region was specified with StoreRegion.  The StoreRegion-RestoreRegion commands work in pairs, and RestoreRegion may not be used a 2 <sup>nd</sup> time on the same region because the memory is freed.
PutBitMap	CMD=51 SEQ=index data= xyy <sub>0</sub> (top left corner)	The bit map loaded with the LoadBitMap is drawn at upper-left corner coordinate xyy <sub>0</sub> .
ScrollRegion	CMD=52 FXN= 1 up 2 down 3 right 4 left data= (int)n, xyy <sub>0</sub> xyy <sub>1</sub>	Data: (int)number of pixels to scroll, followed by rectangular region (exclusive). See “The way the SCROLL command works”.

### **Text Commands**

<b>Command</b>	<b>Stream</b>	<b>Note</b>
SetFont	CMD=60 SEQ=index	Command ignored and error flag set if index not valid.
PutText	CMD=63 data= xyy <sub>0</sub> followed by ASCIIZ string.	Coordinate is at upper-left corner of text.
SetTextDir	CMD=64 FXN= 0 rightward 1 leftward 2 upward 3 downward	The user must specify the font appropriate to the direction. When drawing a text line, the program commences at the specified point, and draws text in the specified direction. If direction = 1, a string “1234” will be printed as “4321”.

### **Miscellaneous Commands**

<b>Command</b>	<b>Stream</b>	<b>Note</b>
BackLight	CMD=70 FXN= 0 light off FXN= 1 light on	The Graphics Engine initializes the back-light to on.

Command	Stream	Note
SetContrast	CMD=71 data = (byte)value	Contrast values range from 0 to 63. A higher number reduces the contrast.
Beep	CMD=72 data = (int)duration	data = duration in milliseconds. 0 = default value (100 msec)
SetCellActive	CMD=74 FXN=Cell index data=(byte)flags	<b>Flags:</b> b0 set = Cell-hit sends unsolicited response to host. b1 set = Beep on cell-hit.  all bits 0 = set cell inactive  Cells are used internally if the Virtual Keyboard is active.
DefineButton  (defines the button but does not display it)  A button consists of an integral number of cells, placed on a cell boundary.  See Figure 6 for the cell numbering scheme.	CMD=75 FXN=flags data	<b>Flags:</b> b0 set = Button-hit sends unsolicited response to host. b1 set = Beep on button-hit. b2 set = Frame button b3 set = Rounded corners <sup>7</sup> b4 set = Invert when button is pushed; normal when button released. b5 set = Check box is normally inverted b6 set = Button text follows <sup>8</sup> b7 set = Button bitmap follows <sup>9</sup> <b>Data bytes</b> in the following order: <i>first two bytes:</i> the buttonId <i>next:</i> index of upper left corner cell <i>next:</i> index of lower right corner cell <i>next if b7 set:</i> bitmap index <i>next if b6 set:</i> ASCIIZ button text
DeleteButton	CMD=76 data=(int)buttonId	<b>Data:</b> the buttonId to be deleted from the button list.
DisplayButton	CMD=77 data=(int)buttonId	The engine saves the area underneath a button when it is displayed, restores it when it is removed. Button is enabled.
RemoveButton	CMD=78 data=(int)buttonId	Disables the button and restores the background. Opposite of DisplayButton.
DisableButton	CMD=79 FXN=0 data=(int)buttonId	The button is disabled (no changes in display).

<sup>7</sup> Requires b2 to be set.

<sup>8</sup> The text is centered in the button, subject to text direction. Include '\n' (line feed) character(s) for multi-line text. If any text line won't fit in the button using the default medium (2) font, it is displayed using the default small (1) font. It is thus possible to mix font sizes in buttons with multi-line text.

<sup>9</sup> The bit map is drawn starting at the top-left-hand corner.



<b>Command</b>	<b>Stream</b>	<b>Note</b>
EnableButton	CMD=79 FXN=1 data=(int)buttonId	Re-enable the button (no changes in display). Opposite of DisableButton.
DisableButtons Area	CMD=79 FXN=2 data	Disable all buttons that are at least partially in the designated touch-screen area (no changes in display).  <b>Data bytes</b> in the following order: <i>1st byte:</i> index of upper left corner cell <i>2nd byte:</i> index of lower right corner cell
EnableButtons Area	CMD=79 FXN=3 data	Enables all buttons that are at least partially in the designated touch screen area which were previously disabled (no changes in display). Opposite of DisableButtonsArea.  <b>Data bytes</b> in the following order: <i>1st byte:</i> index of upper left corner cell <i>2nd byte:</i> index of lower right corner cell
LinkCellTo Macro	CMD=80 FXN= 1 link FXN= 0 unlink data= (int)cell index, (int)macro index	After this command is executed, pushing the specified cell will cause the macro to be played.
LinkButtonTo Macro	CMD=81 FXN= 1 link FXN= 0 unlink data= (int)button index, (int)macro index	After this command is executed, pushing the specified button will cause the macro to be played. A button takes precedence over a cell. Set macro index = 0 for unlink.
StopMacroLoop	CMD=82	Halt macro loop-back. and the execution of the macro itself.
SetString	CMD=84 data = ASCIIZ string.	Load a string into the “default value” buffer. This text is displayed in the Text Entry line of the Virtual Keyboard if the latter is active. Setting a NUL string will clear the line.  If the Virtual Keyboard is not active, the text is stored and displayed when it becomes active.
SetLongInteger	CMD=85 data=(long)integer	Load a long integer into the “default value” buffer. This is converted to ASCII and displayed as in “SetString.”
SetFloat	CMD=86 data=4-byte float	Load a float into the “default value” buffer. This is converted to ASCII and displayed as in “SetString.”

<b>Command</b>	<b>Stream</b>	<b>Note</b>
SetChar	CMD=87 data=char	Append a single character to the “default value” buffer. This is displayed as in “SetString”.
SetTod	CMD=88 data=struct tm	Load the date and time of day and set the Real Time Clock accordingly.
Remove KeyboardMacro	CMD=89 FXN=0	Removes the keyboard macro shown in Figure 3.
PlayKeyboard Macro	CMD=89 FXN=1 SEQ=flags data	<p>Displays the keyboard macro shown in Figure 3.</p> <p><b>Flags:</b></p> <p>b0 set: ‘Enter’ sends unsolicited response to host. This response is in the same form as the SendString command (terminated by CR, NUL).</p> <p>b1 set: Remove keyboard when ‘Enter’ pushed.</p> <p>b2 set: Password mode (echo stars)</p> <p>b3-b4= data entry type</p> <p>00 = string (any entry is OK)</p> <p>01 = long int (check limits)</p> <p>10 = float (check limits)</p> <p>b5 set: not used</p> <p>b6 set: data entry subject to low limit</p> <p>b7 set: data entry subject to high limit</p> <p><b>Data:</b><sup>10</sup></p> <p>(byte)not used (must be supplied)</p> <p>(union)(float or long int)low limit</p> <p>(union)(float or long int)high limit</p> <p>(byte)Prompt Line ASCII string, NUL if none</p>

### **Scripting Commands**

<b>Command</b>	<b>Stream</b>	<b>Note</b>
DrawMacro	CMD=100 FXN,SEQ=ID	This command executes the macro specified by macroId.
ShiftDrawMacro	CMD=101 FXN,SEQ=ID data=xyyy offset	The x,y coordinates of all script commands in the macro are offset by xyyy.
DeleteMacro	CMD=102 FXN,SEQ=ID	Deletes the macro specified by macroId.

<sup>10</sup> Some or all of the data may remain unused. Specify a 0 of appropriate length if not used.

<b>Command</b>	<b>Stream</b>	<b>Note</b>
DelayPlay	CMD=105 data=(long int) delay in msec	Insert a time delay of nnnn milliseconds into the command execution stream. Effective because commands are queued.
LoopMacro	CMD=106	Loop back mode to beginning of macro.

***New (Special) Commands***

<b>Command</b>	<b>Stream</b>	<b>Note</b>
DisplayButtonQ	CMD=107 FXN= 1 normal FXN= 2 inverted data=(int)buttonId	When FXN = 1, the button display is normal; when FXN = 2, the button display is inverted (inverse video). The engine saves the area underneath a button when it is displayed, restores it when it is removed. Button is enabled.
BufferDisplay Enable	CMD=108 FXN= 0 disable FXN= 1 enable	When FXN = 0, the screen buffer lock count is incremented to disable display updates; when FXN = 1 the screen buffer lock count is decremented, and when the lock count reaches zero it enables display updates.

## Appendix I Explanatory Notes

### The way Macros work

Start the macro with a `START_MACRO` command. This command is very simple and all you specify is the command and macroId.

Continue the macro with any command listed in the table in Appendix III. Commands, which are not in the table, are rejected. There is a maximum of 1000 bytes in a macro, including the `START_MACRO` and `END_MACRO` commands. The software stuffs the commands into the macro it's building and gives an error 9 if storage is exceeded. While checking the length limit, it keeps the space for `END_MACRO` in reserve, so that you can issue an `END_MACRO` if you get error 9. Since a macro can be called up from inside a macro, this is not a problem. You can also start all over by issuing a `DELETE_MACRO`.

End the macro with a `END_MACRO` command. Anything between the `START_MACRO` and `END_MACRO` commands is considered to be stuff which goes into the macro

---

*Caution: Make sure not to use a `START_MACRO` without an `END_MACRO`.*

---

Play the macro with a `DRAW_MACRO` command. If a macro is playing, attempts to send commands other than `STOP_MACRO_LOOP` result in error 14.

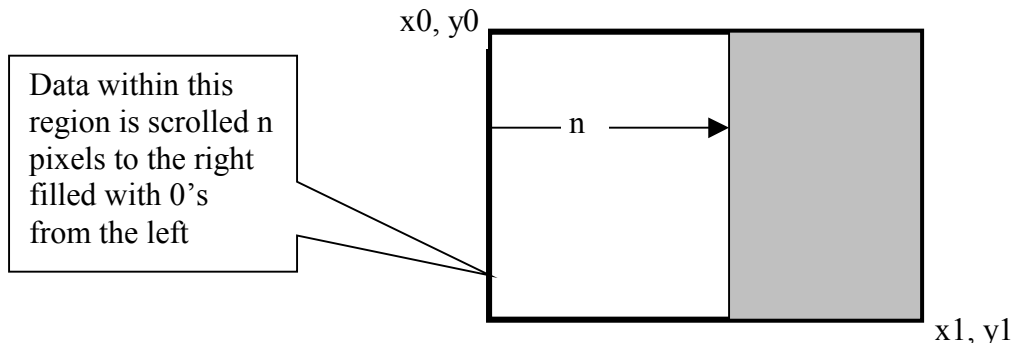
### Inclusive vs Exclusive limits

Inclusive limits apply if both endpoints are included. For example, in the `RECTANGLE` command, if  $x_0 = 40$ , and  $x_1 = 80$ , then the number of columns is 41 (counting the limits).

Exclusive limits apply if the right endpoint is excluded. For example, in the `SCROLL_REGION` command, if  $x_0 = 40$ , and  $x_1 = 80$ , then the number of columns scrolled is 40 (counting the  $x_0$  limit, but not counting the  $x_1$  limit).

### The way the SCROLL command works

The `SCROLL` command specifies a rectangular region and the number of pixels to scroll within that region. For example, to scroll a region bounded by  $x_0, y_0$  and  $x_1, y_1$   $n$  pixels to the right



The SCROLL command is restricted to byte-aligned boundaries in the x direction. That is to say, that  $x_0$  and the width ( $x_1 - x_0$ ) must be evenly divisible by 8. Limits are inclusive, so a specification of, let's say,  $x_0=40$  and  $x_1=80$  scrolls 40 bytes from 40 to 79.

### ***The way the XOR brush works***

This brush works slightly differently, depending on whether a graphic figure (line, circle, etc) or a bitmap is being drawn. In either case, the exclusive-or algorithm works the following way:

$0 \text{ xor } 0 = 0$ (white)	same values, result is white
$0 \text{ xor } 1 = 1$ (black)	different values, result is black
$1 \text{ xor } 0 = 1$ (black)	different values, result is black
$1 \text{ xor } 1 = 0$ (white)	same values, result is white

If a graphic figure is being drawn with a SET brush, it is *specified* to be drawn with black (pixel = 1) at all times regardless of the background (previous contents of the screen). In this case, it will show up if the background is white, but not show up if the background is black.

If a graphic figure is being drawn with an XOR brush, it is *presumed* drawn with black (pixel = 1) and inverts the previous contents of the screen. In this case, it will show up as black if the background is white, and white if the background is black.

If a bitmap is drawn with an XOR brush, then the *contents of the bitmap* are XORed with the previous contents of the screen. Thus, if a bitmap is drawn on top of the same bitmap, it will make the image “disappear”, i.e. go all white ( $0 \text{ xor } 0 = 0$ ,  $1 \text{ xor } 1 = 0$ ).

The best way to invert a bitmap is to draw it on top of a previously drawn bitmap that has all 1's ( $1 \text{ xor } 0 = 1$ ,  $1 \text{ xor } 1 = 0$ ).

## Appendix II Response Formats

The message returned by OP7100GE in response to a command takes one of several formats:

1. Most commands return 5 bytes. The bytes returned are as follows:
  - LEN = 5
  - CMD = same as in command
  - FXN = ACK (0x06)
  - SEQ = lastStatus;
2. If there is an error:
  - LEN = 6
  - CMD = same
  - FXN = NAK (0x15)
  - SEQ = 0x80 (an error occurred)
  - data[0] = the error
3. Some commands (SEND\_STRING, etc) return data:
  - LEN = 5 + number of data bytes
  - CMD = same
  - FXN = ACK (0x06)
  - SEQ = lastStatus
  - data = the data

In any, case, lastStatus is cleared after the response and will be zero at the next response (unless an event happened before then).

### Appendix III Maximum values

Maximum (and in some cases minimum) values are imposed on some of the program parameters:

Parameter	Min/Max values	Comments
Display Queue size	100 entries max	When commands are decoded, the display commands are passed to the display task via the Display Que. Since the transmission time for a command is much shorter than the display time, the queue can become full, resulting in error 7.
Macro Queue size	50 entries max	DrawMacro commands are queued in the Macro Queue in order to be able to specify the display of several macros. Error 7 also occurs if the macro queue becomes full.
fontId	1 – 20 can be specified 4 – 20 can be loaded	A total of 20 fonts can be accommodated: fonts 1 through 3 are default fonts and cannot be loaded
macroId	1 – 32767	
bitmapId	1 – 250	251 – 255 are used internally
buttonId	1 – 32725	32726 – 32767 are used internally
Prompt string	0 – 24	Limited by the length of the line in the Virtual Keyboard (Landscape mode)
Longest allowable macro size	1000 bytes max	Macros can be nested to overcome this limitation
Polygon size	32	Maximum number of vertices in a polygon

## Appendix IV Commands, which can be included in macro

Not all commands can be included in a macro. The following commands are allowed:

BACK_LIGHT	REMOVE_BUTTON
BEEP	RESTORE_REGION
BLANK_REGION	SCROLL_REGION
BUFFER_DISPLAY_ENABLE	SEND_CHAR
CIRCLE	SEND_FLOAT
CLEAR_SCREEN	SEND_LAST_PUSH
CLEAR_STRING_BUFF	SEND_LONG_INT
DELAY_PLAY	SEND_STATUS
DIS_ENA_BUTTON	SEND_STRING
DISPLAY_BUTTON	SEND_TOD
DISPLAY_BUTTON_Q	SET_BRUSH_TYPE
DISABLE_CELLS	SET_CELL_ACTIVE
DRAW_MACRO	SET_CHAR
ENABLE_CELLS	SET_CONTRAST
INVERT_REGION	SET_CURSOR
LINE	SET_CURSOR_MODE
LINK_BUTTON_TOMACRO	SET_FLOAT
LINK_CELL_TOMACRO	SET_FONT
LOOP_MACRO	SET_LONG_INTEGER
NULL_LOOP	SET_STRING
PIXEL	SET_TEXT_DIR
PLAY_KEYBOARD_MACRO	SET_TOD
POLYGON	SHIFT_DRAWMACRO
PUT_BIT_MAP	STOP_MACRO_LOOP
PUT_TEXT	STORE_REGION
RECTANGLE	



## Command Index

BackLight, 15  
Beep, 16  
BlankRegion, 14  
BufferDisplayEnable, 19  
Circle, 14  
ClearScreen, 13  
ClearStringBuffer, 12  
DefineButton, 16  
DelayPlay, 19  
DeleteBitMap, 12  
DeleteButton, 16  
DeleteFont, 12  
DeleteMacro, 18  
DisableButton, 16  
DisableButtonsArea, 17  
DisableCells, 13  
DisplayButton, 16  
DisplayButtonQ, 19  
DrawMacro, 18  
EnableButton, 17  
EnableButtonsArea, 17  
EnableCells, 13  
EndMacro, 13  
Init, 11  
InvertRegion, 14  
Line, 14  
LinkButtonToMacro, 17  
LinkCellToMacro, 17  
LoadBitMap, 12  
LoadFont, 12  
LoopMacro, 19  
Pixel, 13  
PlayKeyboardMacro, 18  
Polygon, 14  
PutBitMap, 15  
PutText, 15  
Rectangle, 14  
RemoveButton, 16  
RestoreRegion, 15  
ScrollRegion, 15  
SendChar, 12  
SendFloat, 12  
SendLastPush, 11  
SendLongInteger, 12  
SendStatus, 11  
SendString, 12  
SendTod, 12  
SetBrushType, 13  
SetCellActive, 16  
SetChar, 18  
SetContrast, 16  
SetFloat, 17  
SetFont, 15  
SetLongInteger, 17  
SetString, 17  
SetTextDir, 15  
ShiftDrawMacro, 18  
StartMacro, 13  
StopMacroLoop, 17  
StoreRegion, 14  
SuperReset, 13