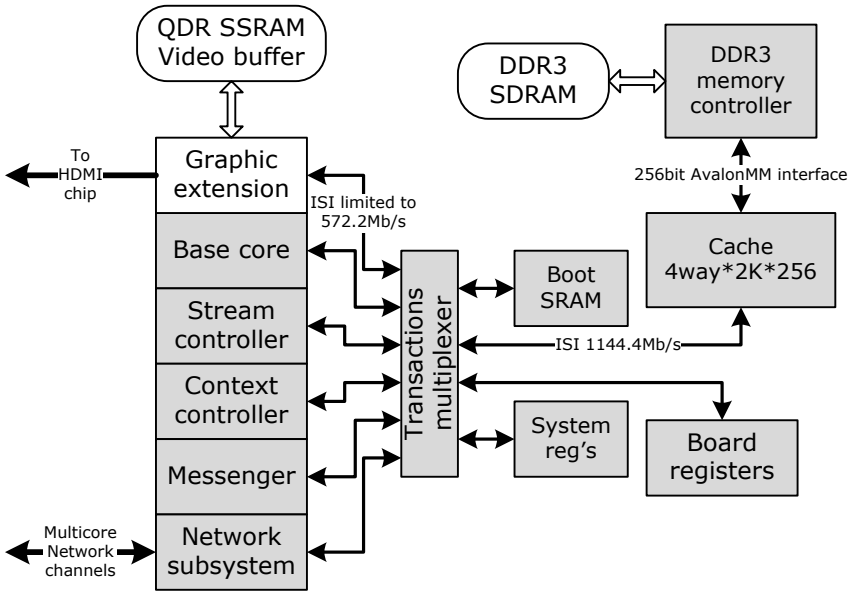
The architecture of the graphic extension.

Table of contents

Graphic extension unit architecture.

Connection of the graphic extension to the Core.

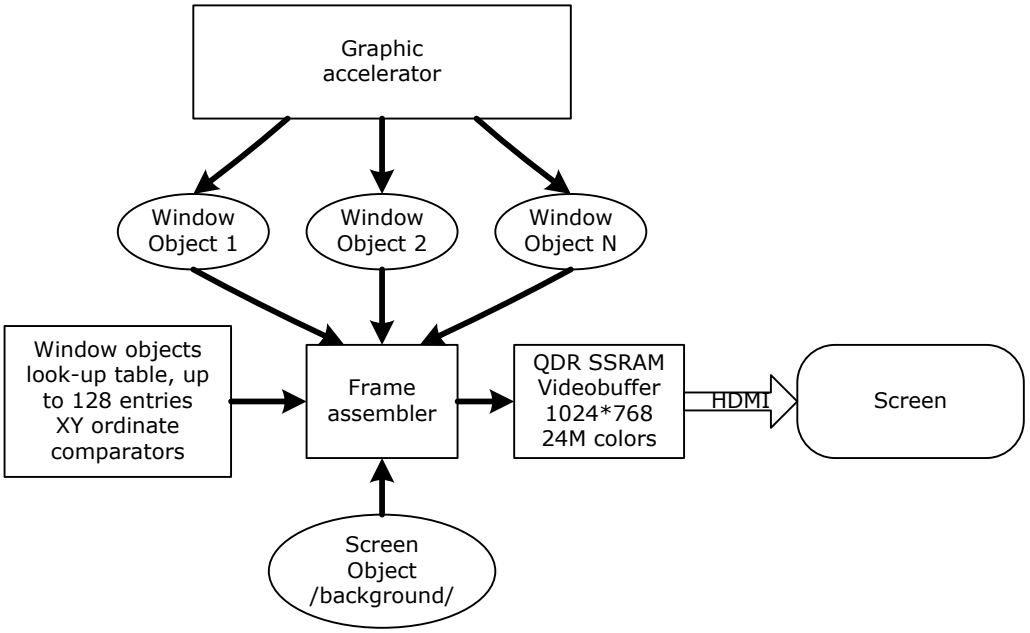
The figure below shows the general configuration of a processor system with a graphics extension. Gray shading indicates the basic blocks that are always present in the Core, in any configuration. QDR SSRAM and DDR3 SDRAM are located on the Stratix IV GX FPGA Development Board.



The graphics extension memory access channel has a forced bandwidth limitation of 2 times the maximum. This was done so that the graphic extension could not block the work of other processor modules with its transactions.

Graphic extension structure.

The figure below shows the generalized structure of the graphical extension. The graphical extension works with data from several objects located in the main RAM.



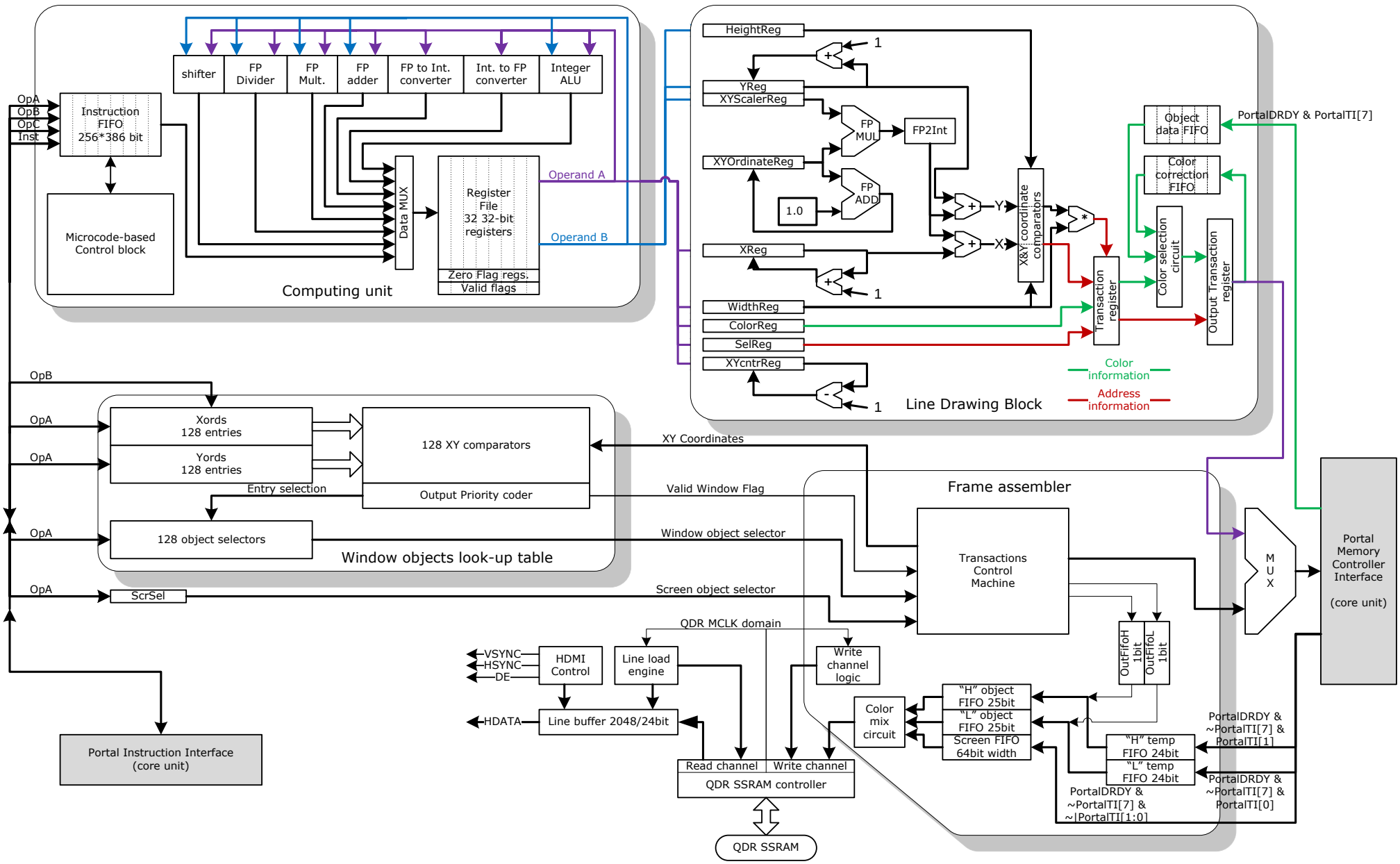
The screen object contains data that represents the background image and is always displayed on the screen at points that are not occupied by any window. The screen object must be 1024*768*4=3145728 bytes long.

Window objects are used by processes to display graphical information. Window objects can be of different lengths, depending on the size of the windows. A single process can own one or more window objects.

The frame assembler assembles the image and places it in the frame buffer. The assembly is carried out taking into account which window data should be used to display a specific pixel. To do this, a lookup table of 128 entry points is used. Each point describes the parameters of one window - the start and end of the window in the X coordinate, the start and end in the Y coordinate, and the window object selector. If several windows overlap on the same pixel, then the window parameter entry number in the lookup table determines the currently visible window.

The graphics accelerator draws such simple shapes as lines, rectangles filled with one color, triangles and ellipses. For ellipses, you can separately set the color of the points on the border of the ellipse and the color of the fill inside the ellipse. To draw a figure, the accelerator needs to specify in the appropriate instructions the window object selector, the window parameters /width and height/ and the figure parameters /output coordinates, dimensions, colors/.

Architecture in detail.


OpA
OpB
OpC
Inst
Instruction FIFO 256*386 bit
Microcode-based Control block
shifter
FP Divider
FP Mult.
FP adder
FP to Int. converter
Int. to FP converter
Integer ALU
Data MUX
Register File 32 32-bit registers
Zero Flag regs.
Valid flags
Operand A
Operand B
Computing unit
HeightReg
YReg
XYScalerReg
XYOrdinateReg
1.0
XReg
WidthReg
ColorReg
SelReg
XYcntrReg
FP MUL
FP ADD
FP2Int
X&Y coordinate comparatrs
Transaction register
Object data FIFO
Color correction FIFO
Color selection circuit
Output Transaction register
PortalDRDY & PortalTI[7]
Color information
Address information
Line Drawing Block
OpB
OpA
Xords 128 entries
Yords 128 entries
Entry selection
128 XY comparators
Output Priority coder
128 object selectors
Window objects look-up table
ScrSel
XY Coordinates
Valid Window Flag
Window object selector
Screen object selector
Frame assembler
Transactions Control Machine
QDR MCLK domain
VSYNC
HSYNC
DE
HDATA
HDMI Control
Line load engine
Line buffer 2048/24bit
Write channel logic
Color mix circuit
"H" object FIFO 25bit
"L" object FIFO 25bit
Screen FIFO 64bit width
OutFifoH 1bit
OutFifoL 1bit
"H" temp FIFO 24bit
"L" temp FIFO 24bit
PortalDRDY & ~PortalTI[7] & PortalTI[1]
PortalDRDY & ~PortalTI[7] & PortalTI[0]
PortalDRDY & ~PortalTI[7] & ~|PortalTI[1:0]
Read channel
Write channel
QDR SSRAM controller
QDR SSRAM
M U X
Portal Memory Controller Interface
(core unit)
Portal Instruction Interface
(core unit)

The work of the frame collector is controlled by the Transaction Control Machine /TCM/. TCM reads data from the screen object and data from the window objects. In parallel with reading data from the screen object, TCM generates the horizontal and vertical coordinates of the point for the Window objects look-up table. The look-up table contains 128 descriptors that describe the position and size of the window into which the process displays its graphical information.

71 48	36	24	12	0
Window Object Selector 127	Y bottom 127	Y top 127	X right 127	X left 127
⋮				
Window Object Selector 1	Y bottom 1	Y top 1	X right 1	X left 1
Window Object Selector 0	Y bottom 0	Y top 0	X right 0	X left 0

In addition to the window sizes, each set contains an object selector in which graphic information is located. The [X left, Y top] and [X right, Y bottom] pairs determine the size and location of the window on the screen. Information from the window descriptors is fed simultaneously to 128 comparators, which check that the point coordinates generated in the TCM fall into the corresponding window. The outputs of 128 comparators go to a priority encoder, which selects the comparator channel with the highest index. For example, if the coordinates of a point fall into windows whose descriptors are located in positions 3 and 10 of the table, then the priority encoder will select the descriptor with index 10. Window objects look-up table, when it detects that the coordinates of a point fall into any window, calculates the displacement of the point in the object and returns the window object selector and the offset within the object to the TCM. Simultaneously, two pixels are located, even and odd. If both pixels fall into the same window object, then the controller reads a 64-bit word from memory. When the even and odd points hit different window objects, then the TCM will be forced to perform 2 separate transactions reading 32-bit values from different window objects.

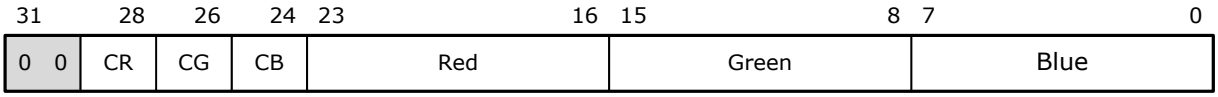
2 1-bit queues OutFifoL and OutFifoH are filled with 1-bit flags as it is determined which object the even and odd pixels belong to. If the pixel does not fall into any of the window objects and needs to be rendered from the screen object, then bit 0 is written to the corresponding queue; if the pixel belongs to the window object, the TCM places 1 into the corresponding queue at the time the read transaction is sent to the portal.

At the output of the OutFifoL and OutFifoH queues, 1-bit flags of the presence of a data read request sent to the memory subsystem through the portal appear. If the output of any queue is set to 0, then this 0 is immediately written to the 25th bit of the "H"ObjectFifo or "L"ObjectFifo And this corresponds to the situation when there is no data for the pixel from any window object. If the output of the OutFifoL and OutFifoH queues is set to 1, then this flag will not be removed from the queue until the corresponding queue "L"TempFifo or "H"TempFifo has accumulated at least one value read from memory. Then the pixel value from "L"TempFifo or "H"TempFifo followed by a 1 in the 25th bit is put into the "L"ObjectFifo or "H"ObjectFifo queue.

ScreenFIFO contains data read from the screen object.

Formation of the pixels color.

When data is present in all three queues: "L"ObjectFifo, "H"ObjectFifo and ScreenFIFO, 2 pixels are written to the QDR SSRAM frame buffer. Before recording, the resulting color of the point is formed depending on the color control bits in a 32-bit word read from the screen buffer. The figure below shows the data format used for the screen buffer.



The lower 24 bits are used for color encoding: [23:16]==R, [15:8]==G, [7:0]==B. Bits [31:30] – not used. Bits [29:24] contain 3 pairs to control the formation mode of each of the three colors separately. Bit pairs determine the results for the R, G, B channels in accordance with the table:

CR[1:0] CG[1:0] CB[1:0]	***Color Function***
0	Color=CW
1	Color=CW xor CS
2	Color=CW + CS
3	Color=CS

CW – color value from window object

CS – color value from screen object

Graphics extension transaction tag.

PortalTI[7:0]	***Data receiver***
0XXXXX00	ScreenFIFO<=PortalDI[63:0]
0XXXXX01	"L"ObjectFifo<=PortalDI[31:0]
0XXXXX10	"H"ObjectFifo<=PortalDI[31:0]
0XXXXX11	"L"ObjectFifo<=PortalDI[31:0], "H"ObjectFifo<=PortalDI[63:32]
1XXXXXXX	ObjectDataFIFO<=PortalDI[31:0]

Graphics accelerator architecture.

The graphics accelerator can be divided into two blocks: a computing unit and a line drawing unit. The Computing Unit is used to calculate the coordinates of the start and end points of the lines, as well as the slope coefficient of the line. The line drawing block fills the object with data that forms the line and is capable of forming a transaction for writing data to memory at each clock cycle. The block draws horizontal, vertical and oblique lines.

The computing unit operates under the control of a microprogram control unit, which has microcode procedures for executing instructions for drawing lines, rectangles, triangles and ellipses.

A set of 32 32-bit general purpose registers is designed to store data used in calculating the coordinates of the beginning and end of lines. Each register has 2 flag bits associated with it. The register is ready to be used in a new microinstruction and the zero result flag. The zero result flag is set each time the register is written to. The

register ready bit is reset to 0 when a microinstruction using the register as a result destination is sent for execution. Different microinstructions have different execution times, and register ready bits allow new microinstructions to be executed as their source operands are ready.

The computing unit contains 8 data processing channels that can send results to registers.

1. An integer ALU that performs addition, subtraction, logical AND, OR, XOR, register-to-register transfers, and constant loading.
2. Converts numbers from integer format to single precision floating point format.
3. Converts numbers from floating point format to integer format.
4. A floating point adder that performs additions and subtractions.
5. Floating point multiplier.
6. Floating point divider.
7. A variable-bit shifter that performs logical right/left shift and arithmetic right shift operations.
8. The eighth channel from which data can flow into general purpose registers is the instruction queue. The instruction queue contains 384 bits of instruction parameters /12 32-bit words/ and 2 bits of instruction code. The instruction code specifies the drawing of 0-line, 1-rectangle, 2-triangle and 3-ellipse.

The computing unit calculates the coordinates of the beginning and end of the line, determines the mode of drawing the line - vertically or horizontally, and determines the slope coefficient of the line. The calculation results are transferred to the line drawing block in the form of the following parameters:

1. Coordinates of the beginning of the line.
2. Drawing mode flag – horizontal or vertical.
3. Number of points in a line.
4. Line color.
5. Width and height of the window.
6. Selector of the object.
7. Line slope coefficient /optional/.

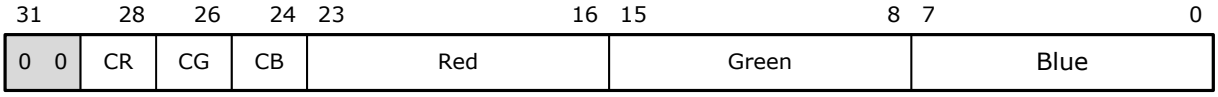
Lines can be drawn in two modes - orthogonal or slanted. Orthogonal mode draws horizontal or vertical lines. Tilt mode uses a pipeline to calculate the increment of the X-axis or Y-axis coordinates depending on the progress along the Y-axis or X-axis, respectively. The resulting offset value is converted into an integer value and summed with the base X or Y coordinate to form the full coordinate of the next point. The full coordinates of a point are converted to an offset within the object by multiplying the Y coordinate by the window width value, adding the X coordinate value to the product, and then shifting it 2 bits to the left.

The coordinates of the beginning of the line are written to the Xreg, Yreg registers. During line drawing, XReg is incremented by 1 if the drawing mode is set to horizontal with or without line slant. The YReg register is incremented by 1 if the drawing mode is set to vertical.

The WidthReg and HeightReg registers set the window dimensions and are used to check the generated point coordinates to see if they fall inside the window. If the coordinates of a point go beyond the boundaries of the window, the transaction is blocked. WidthReg is also used to form the offset of a memory cell by multiplying by the value of the point's Y axis coordinate.

The XYScalerReg and XYOrdinateReg registers are used only if the line output mode is not orthogonal. When the line drawing procedure is started, the XYOrdinateReg register is reset to zero and is incremented by 1.0 with each new point. The XYOrdinateReg value is multiplied by the XYScalerReg value to obtain the X offset if the line output mode is vertical and the Y offset if the line output mode is horizontal. The product is transformed to an integer and added to the XReg or YReg value.

ColorReg is a 30-bit register containing the color with which the line is drawn. Its format is shown in the figure below.



The lower 24 bits are the actual pixel color or the value for modifying the pixel color in accordance with the modes specified in bits [29:24]. If the state of these bits is zero, then the new pixel value is written to the object. If not zero, then the previous pixel value must be read in order to modify it as needed. In this case, the value that modifies the pixel and the logical address of the pixel are entered into the Color Correction FIFO queue, and the Object Data Fifo queue receives values read from memory. And when there is valid data in both queues, then the resulting color is generated and the pixel is written to the object. The resulting color is formed for each color channel individually in accordance with the table:

CR[1:0] CG[1:0] CB[1:0]	***Color Function***
0	Color=ColorReg
1	Color=ColorReg xor CO
2	Color=ColorReg + CO
3	Color=CO

ColorReg – color value from ColorReg

CO – color value from memory object

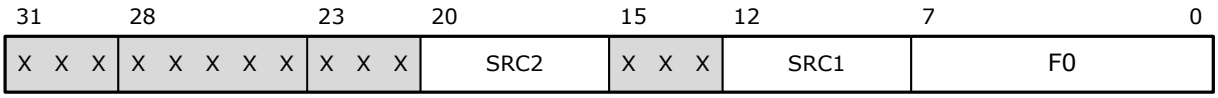
Graphics extension instruction set.

SWOP - Set Window Object Parameter

Mnemonic:

SWOP src2,src1

Format:



Description:

The instruction writes a 32-bit parameter from the SRC1 register to the window object parameter table. The table cell is addressed by the contents of the SRC2 register. For example, if the SRC2 register contains the value 0Dh, then this register addresses the Y parameters for the 3rd window parameter descriptor in the table.

Example:

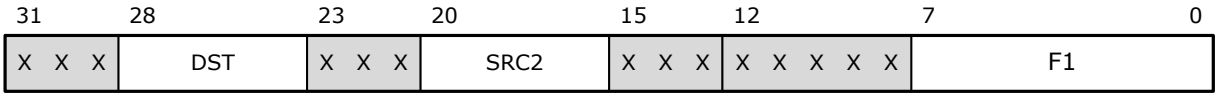
```
swop    r20,r0
```

GWOP – Get Window Object Parameter

Mnemonic:

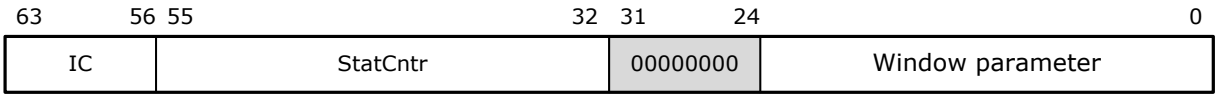
GWOP dst,src2

Format:



Description:

The instruction reads a table cell of window objects, as well as some state parameters of the graphical extension, and places the result in a general-purpose register. The destination register of the result is indicated in the DST field, and the SRC2 field indicates the register containing the address of the window object table cell. The result format is shown in the figure below.



The lower 24 bits are the window parameter from the window object table.

The StatCntr statistics counter field provides information about how many average wait cycles are required per transaction between the graphics extension and main memory. The averaging is performed on a rolling sample of 4096 transactions, so to get the average of the wait cycles per transaction, you need to shift the StatCntr value 12 bits to the right.

The IC field provides information about the number of instructions in the graphics accelerator instruction queue. This information can be used to determine whether a new instruction can be sent to the accelerator.

Example:

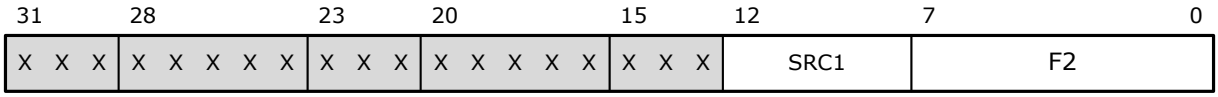
```
gwop    r15,r30
```

SSO – Set Screen Object

Mnemonic:

SSO src1

Format:



Description:

The contents of the general purpose register specified in the SRC1 field are written to the screen object selector register /ScrSel register/. The graphics extension builds the frame only if the value of the screen object selector is non-zero. If ScrSel is set to zero, frame assembly work stops, but the graphics accelerator can continue its work.

Example:

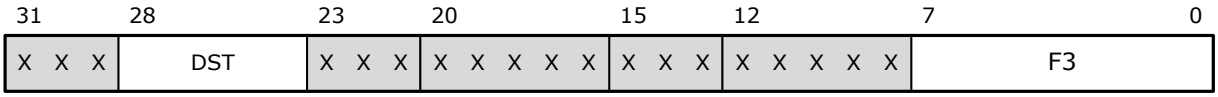
sso r3

GSO – Get Screen Object

Mnemonic:

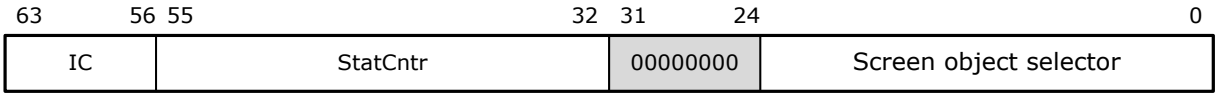
GSO dst

Format:



Description:

The instruction reads the ScrSel register and status information, placing the result in the register specified in the dst field. Instruction result format:



Bits [23:0] contain the screen object selector. Since bits [31:24] are zero, the selector always describes an object located in the core's local memory. The screen object cannot reside in the memory of another core in a multi-core system, only in local memory.

Bits [55:32] – statistics on the number of clock cycles spent on executing a transaction between main memory and the graphics extension.

Bits [63:56] – the number of instructions in the graphics accelerator queue.

Example:

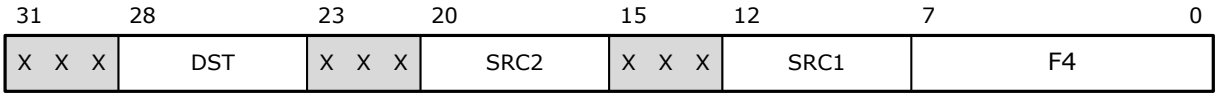
gso r22

DRAWL – Draw Line

Mnemonic:

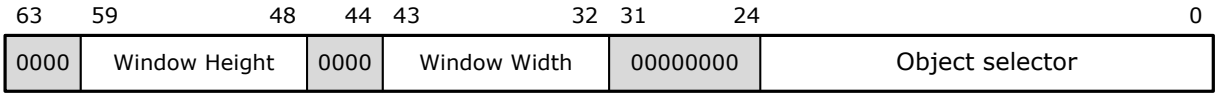
DRAWL src1,src2,dst

Format:



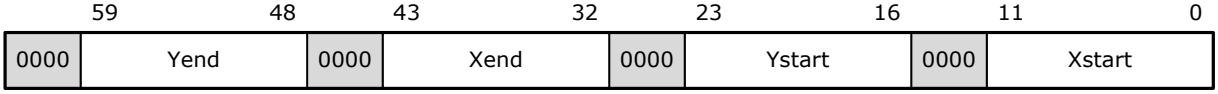
Description:

Instructions for drawing a line using a graphics accelerator. The src1 register contains the object selector and the window width and height parameters. The size of the object must be equal to or greater than WindowWidth*WindowHeight*4 bytes.

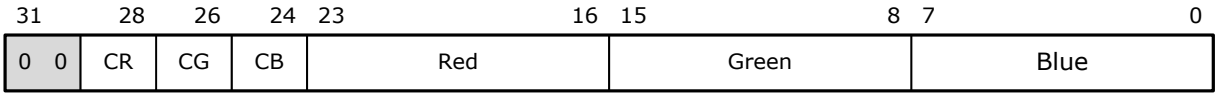


The two parameters Window Width and Window Height indicate the size of the window.

The src2 register contains the coordinates of the starting and ending points of the line.



The dst register contains the line color and control bits for the line color generation mode.



Example:

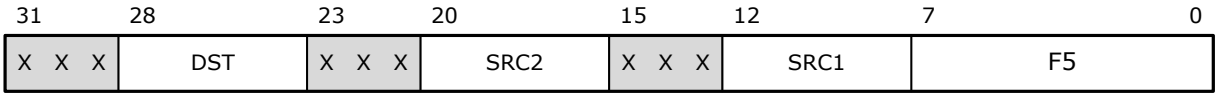
drawl r6,r19,r17

DRAWR – Draw Rectangle

Mnemonic:

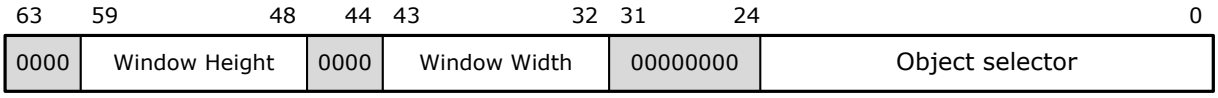
DRAWR src1,src2,dst

Format:

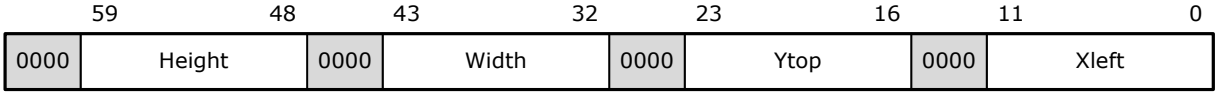


Description:

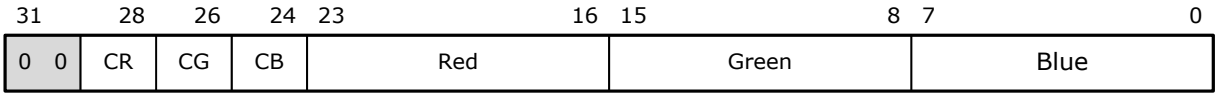
Instructions for filling the rectangle. The src1 register contains the object selector where to fill and size the window.



The src2 register contains the coordinates of the upper left corner of the rectangle, its width and height.



The dst register contains the fill color and the resulting color control bits.



Example:

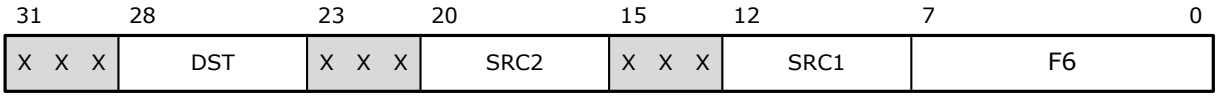
drawr r6,r21,r23

DRAWT – Draw Triangle

Mnemonic:

DRAWT src1,src2,dst

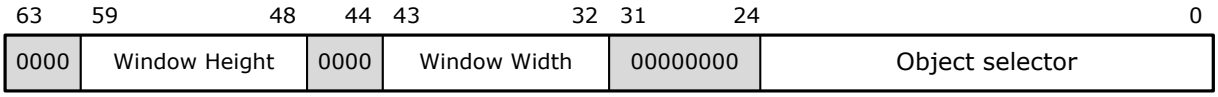
Format:



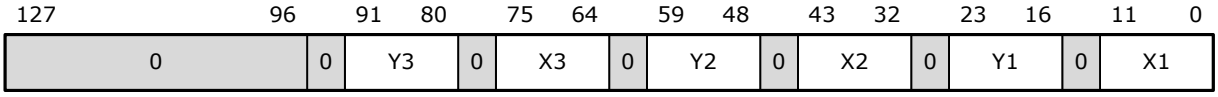
Description:

Instructions for filling a triangle with color.

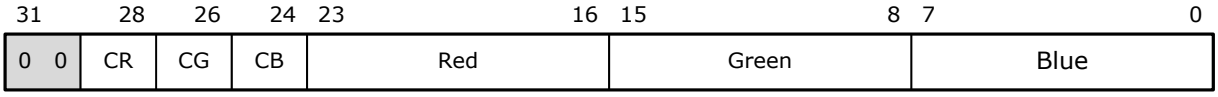
The register specified in the src1 field contains the object selector in which you want to form a filled triangle and the size of the window in this object.



The register specified in the src2 field contains three coordinates of the angles of the triangle.



The register specified in the dst field contains the triangle fill color and color control bits.



Example:

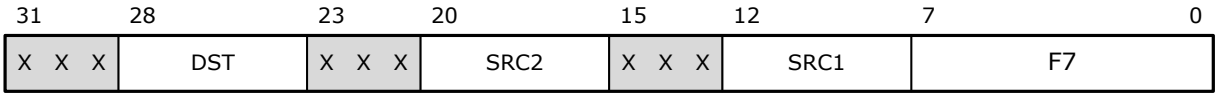
drawt r6,r19,r17

DRAWE – Draw Ellipse

Mnemonic:

DRAWE src1,src2,dst

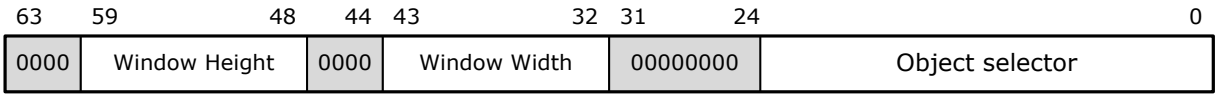
Format:



Description:

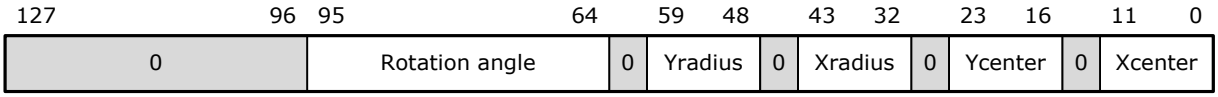
Instructions for drawing an ellipse and filling its internal space with the required color. Using this instruction, you can draw only the outline of an ellipse, or only a completely filled ellipse, or an ellipse with a fill and border of different colors.

The register specified in the src1 field contains the object selector and window dimensions.

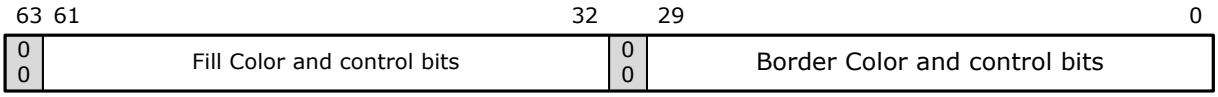


The register specified in the src2 field must contain three parameters:

1. Coordinate of the center of the ellipse.
2. Dimensions of the ellipse along the X and Y axes.
3. The ellipse angle ranges from $-\pi/2$ to $\pi/2$ in single-precision floating-point format.



The register specified in the dst field must contain the color of the ellipse border and its fill color, along with bits that control the mode of formation of the resulting color.



Example:

drawe r6,r23,r20