# rtfGraphicsAccelerator

robfinch<remove>@finitron.ca

## Overview

This is a video graphics display accelerator that enhances system performance. It contains a dedicated character blitter supporting multiple font sizes which allows a simple software interface. It also contains a general purpose blitter component allowing fast graphics transfers and manipulation.

The accelerator interfaces as a slave device with the cpu via a 64-bit bus. It interfaces to the memory system as a bus master using a 128-bit bus.

## History

Initially this accelerator was an all-in-one audio/video controller core. Several components were separated out of the core to leave just the graphics acceleration. Consequently, a large number of registers were removed leaving a somewhat sparsely populated register set.

## Features

16 bpp or 32 bpp color depth

Target area specification up to 65535x65535

graphics command queue

character blitting, variable bitmap font size from 1 x 1 to 32 x 32

point plot / line / rectangle / triangle / curve draw acceleration

general purpose blitter – three sources, one destination

## Documentation Notes

Some registers are illustrated as if they were 32 bits in size for easier readability. The lower half of a register is presented first, followed by the upper half. These are given addresses that are four bytes apart. However, registers in the circuit are really 64-bit.

## Clocks

Two clocks are used, one for the slave interface and second clock for the master interface. The two clocks operate independently.

## Display Format

The accelerator uses a target area which may be a different size than the display resolution determined by the sync controller.

## Color Depth

The controller uses sixteen or thirty-two bits per pixel color depth (RGB444 or RGB888) with extra bits to indicate z-order. The most significant four or eight bits of each color is used to create a z-order number. The z-order number is used in comparison to the z-order of a cursor / sprite to determine whether the pixel appears in front of or behind the sprite.

## Color Planes

The plane a pixel is located on is determined by z-order. Plane #0 is the front plane. Plane #$FF is the rearmost plane.

## Display Memory

The core typically expects a memory of least 64k words by 128 bits wide (1MB) to economize on the pixel format of ZRGB4444 or ZRGB8888 and make it possible for the processor to use the memory in a general-purpose fashion. The controller has a bus master port separate from the slave port used to update the controller’s registers. Typically, memory will be shared between the controller and processor through some sort of bus arbitrator and multiplexor. The controller uses a 128-bit path to memory to maximize the rate of transfer.

Bus transfers are performed using single read / write cycles.

## Horizontal Line Draw

The horizontal line draw procedure used to fill objects will draw all eight pixels of a memory strip in parallel if possible. This makes the horizontal line draw up to eight times faster than other drawing.

# Fonts

The controller features a dedicated text blitter than can handle either fixed or varying width fonts. Multiple fonts may be supported through the use of font tables.

## Font Table

The core supports the use of multiple fonts onscreen at the same time via a font table. The font table is a table of information describing basic characteristics of the font and where to find further font information for a given number of fonts. The font table is in memory and indexed by the font id register to select a font to work with. The font table is a collection of font table entries each of which has the following layout:

Font Table Entry

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Offset | Fields | | | | Use |
| 0 | Address31..0 | | | | Address of character bitmaps |
| 4 | fixed1 | width5 | height5 | ~21 | width and height |
| 8 | Address31..0 | | | | Glyph width table address |
| C | ~32 | | | |  |

Each font table entry is sixteen bytes in size. The font id (register $1E8) is used to index into this table. Setting the font table id register tells the core which font to use. The core then looks up the font information from the table.

The location of the font table in the controller’s memory is specified in the font table address register - register $1E0.

## Glyph Width Table

If the font is a fixed width font then no further table lookup is required, and the font width is determined by the width field in the font table entry. For fonts with characters whose bitmaps vary in width there is an additional table used to describe the width of the bitmap for the character in memory. Thus, variable width character fonts are supported.

## Font Table Address

This register determines where in the controller’s memory the font table is located.

The glyph width table is an array of bytes that specify the character width for each character in the font. The address of the glyph width table is found in the font table entry for the font.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 63 0 | | |  |
| $1E0 | ~32 | Offset31..0 | | font table address |
| $1E8 | ~48 | | Font id16 |  |

## Font ID

This register selects the working font. The core uses this id to determine which entry of the font table to use to lookup font attributes.

# Registers

### Target (screen bitmap) Base

This register determines where in the controller’s memory the target area (bitmap) for the screen display is located. The bitmap base address may be located at any address to allow smooth scrolling the display simply by changing the base address.

|  |  |  |
| --- | --- | --- |
|  | 31 0 |  |
| $3C0 | Offset31..0 | Target bitmap base low |
| $3C4 | ~32 | Target bitmap base high |

### Target Size

This register controls the size of the target bitmap. Note that the target bitmap may be larger than the screen size. The size of the screen is controlled by the sync generator registers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 63 0 | | |  |
| $3D0 | ~32 | Width16 | Height16 | Target Size |

## Command Queue

### Overview

The controller uses a command queue to allow the main processor to continue operating asynchronously to graphics processing. Some graphics operations may require hundreds or thousands of clock cycles to complete. The processor is not stalled while these operations take place. The command queue holding register is written with desired parameters and command code. Then a value is written to the queue trigger register. Writing the queue trigger register causes the holding registers to be placed in a queue. Graphics commands are processed from the queue in first-in first-out order. Note that the holding registers retain their value after being queued so that another command may be queued by modifying only the values that need to change.

The queue has a fixed number of entries 1025 to be exact. More items should not be placed in the queue until there is available queue space. The number of entries currently queue can be read from the queue count register.

### Command Queue Count Register

This register contains the number of entries currently in the command queue. As commands are executed this number will go down. The queue count register should be checked before sending graphics commands.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 63 0 | |  |
| $1D0 | ~48 | Count16 | Command count |

### Command Queue Holding Register

|  |  |  |  |
| --- | --- | --- | --- |
|  | 31 0 | |  |
| $1C0 | Parameter31..0 | | Command parameter |
| $1C4 | ~24 | Command Code8 | Command Code |

### Command Queue Trigger Register

|  |  |  |
| --- | --- | --- |
|  | 63 0 |  |
| $1C8 | ~64 |  |

### Command Register

The command code register specifies which graphics operation to perform. Write to the queue trigger register ($1C8) to queue the command.

### Summary of Graphics Commands

|  |  |  |
| --- | --- | --- |
| Command8 | Operation Performed | Parameters / Set first |
| 0 | Draw character bitmap | X0,Y0 fgcolor, bkcolor, char code |
| 1 | Plot point | X0,Y0 bkcolor |
| 2 | Draw line | x0,y0 x1,y1 bkcolor |
| 3 | Draw filled Rectangle | x0,y0 x1,y1 bkcolor |
| 6 | Draw filled triangle | x0,y0 x1,y1 x2,y2, bkcolor |
| 8 | Draw Bezier curve | x0,y0 x1,y1 x2,y2, bkcolor, fill code |
| 12 | Set pen color | RGB888 value |
| 13 | Set fill color | RGB888 value |
| 14 | Set alpha | 16-bit value |
| 16 | Set X0 | 32-bit fixed point (16,16) value |
| 17 | Set Y0 | 32-bit fixed point (16,16) value |
| 18 | Set Z0 | 32-bit fixed point (16,16) value |
| 19 | Set X1 | 32-bit fixed point (16,16) value |
| 20 | Set Y1 | 32-bit fixed point (16,16) value |
| 21 | Set Z1 | 32-bit fixed point (16,16) value |
| 22 | Set X2 | 32-bit fixed point (16,16) value |
| 23 | Set Y2 | 32-bit fixed point (16,16) value |
| 24 | Set Z2 | 32-bit fixed point (16,16) value |
| 25 | Set Clip X0 | 16-bit value (whole number) |
| 26 | Set Clip Y0 | 16-bit value (whole number) |
| 27 | Set Clip X1 | 16-bit value (whole number) |
| 28 | Set Clip Y1 | 16-bit value (whole number) |
| 29 | Set clip enable / disable | 1 bit value |
| 32 | Set aa transform coefficient | 32-bit fixed point (16,16) value |
| 33 | Set ab transform coefficient | 32-bit fixed point (16,16) value |
| 34 | Set ac transform coefficient | 32-bit fixed point (16,16) value |
| 35 | Set at transform coefficient | 32-bit fixed point (16,16) value |
| 36 | Set ba transform coefficient | 32-bit fixed point (16,16) value |
| 37 | Set bb transform coefficient | 32-bit fixed point (16,16) value |
| 38 | Set bc transform coefficient | 32-bit fixed point (16,16) value |
| 39 | Set bt transform coefficient | 32-bit fixed point (16,16) value |
| 40 | Set ca transform coefficient | 32-bit fixed point (16,16) value |
| 41 | Set cb transform coefficient | 32-bit fixed point (16,16) value |
| 42 | Set cc transform coefficient | 32-bit fixed point (16,16) value |
| 43 | Set ct transform coefficient | 32-bit fixed point (16,16) value |
| 254 | Reset command queue |  |
| 255 | NOP |  |

### Command #0 – Draw Character Bitmap

Command #0 will draw the character specified by the low order 16-bits of the parameter portion of the command holding register at previously set X0, Y0 co-ordinate registers, using a previously set foreground and background color. Bit 31 of the register indicates to update X0 by adding the width of the character bitmap to it. Thus, a string of characters may be written simply by queuing the characters in succession.

### Command #1 – Plot Point

Command #1 will plot a point on the target bitmap using previously set X0, Y0 co-ordinates in the previously set background color. The raster operation used to set the point is determined from bits 4 to 7 of the command parameter.

### Command #2 – Draw Line

Command #2 will draw a line on the target bitmap using the previously set X0, Y0, X1, and Y1 co-ordinates in the previously set fill color. The raster operation used is determined from bits 4 to 7 of the command parameter.

### Command #3 – Draw Filled Rectangle

Command #3 will draw a filled rectangle on the target bitmap using the previously set X0, Y0, X1, and Y1 co-ordinates in the previously set fill color. The raster operation used is determined from bits 4 to 7 of the command parameter.

### Command #6 – Draw Triangle

Command#6 will draw a filled triangle on the target bitmap using the previously set X0, Y0, X1, Y1, and X2, Y2 co-ordinates. The raster operation used is determined from bits 4 to 7 of the command parameter.

### Command#8 – Draw Bezier Curve

Command#8 will draw a filled or unfilled Bezier curve on the target bitmap using previously set co-ordinates. The least significant two bits of the command parameter determine how the curve is filled. The raster operation used is determined from bits 4 to 7 of the command parameter.

### Command #12 – Set Pen Color

This command will set the graphics pen color for subsequent operations using the pen color. The pen color set must be a RGB888 value.

### Command #13 – Set Fill Color

This command will set the fill color for subsequent operations using the fill color. The fill color set must be an RGB888 value.

### Command #14 – Set Alpha

This command sets the alpha value for subsequent operations using an alpha value. The alpha value is a 16-bit number.

### Command #16 – Set X0

This command sets the graphics X0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #17 – Set Y0

This command sets the graphics Y0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #18 – Set Z0

This command sets the graphics Z0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #19 – Set X1

This command sets the graphics X1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #20 – Set Y1

This command sets the graphics Y1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #21 – Set Z1

This command sets the graphics Z1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #22 – Set X2

This command sets the graphics X2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #23 – Set Y2

This command sets the graphics Y2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #24 – Set Z2

This command sets the graphics Z2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #25 – Set ClipX0

This command sets the clipping region X0 co-ordinate. The parameter value is a 16-bit integer (whole number only).

### Command #26 – Set ClipY0

This command sets the clipping region Y0 co-ordinate. The parameter value is a 16-bit integer.

### Command #27 – Set ClipX1

This command sets the clipping region X1 co-ordinate. The parameter value is a 16-bit integer (whole number only).

### Command #28 – Set ClipY1

This command sets the clipping region Y1 co-ordinate. The parameter value is a 16-bit integer.

### Command #29 – Set Clip Enable / Disable

This command determines whether clipping of graphics output is enabled or disabled. If the parameter is 1 clipping is enabled, if the parameter is 0 then clipping is disabled. Note that all graphics are automatically clipped according to the target width and height which cannot be disabled. This setting controls the clip region defined by clipping co-ordinates clipx0, clipy0, clip x1, clip y1.

### Command #254 – Reset Command Queue

This command empties out the command queue. Any command in the queue are not performed.

### Command #255 – NOP

This command is a no-operation.

## Blitter

### Overview

The accelerator has a powerful blitter component which may be used to transfer information in the system’s memory extremely fast. The blitter consists of four DMA channels (A, B, C, and D). A, B, and C are data source channels and D is a data destination channel. The destination channel may be used in a standalone fashion to draw lines or fill areas. Any or all three of the source channels may be active to fetch data to transfer to the destination. A variety of operations between the data fetched by channels A, B, and C are possible including copy and masking operations.

### Pipeline

The blitter features pipelined memory access with a programmable pipeline depth. When pipelining is active the blitter performs burst memory accesses for a given channel until the pipeline queue is full. It then moves onto the next channel. Once all the queues are full then data operations on the values are performed as the destination memory is being written. This provides the most efficient use of memory bandwidth.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 31 | | | | 0 |  |
| $100 | Address31..0 | | | | | Channel A address low |
| $104 | ~32 | | | | | Channel A address high |
| $108 | Offset31..0 | | | | | Channel A modulo low |
| $10C | ~32 | | | | | Channel A modulo high |
| $110 | Count32 | | | | | Channel A Count |
| $114 | ~32 | | | | |  |
| $120 | Address31..0 | | | | | Channel B address low |
| $124 | ~32 | | | | | Channel B address high |
| $128 | Offset31..0 | | | | | Channel B modulo low |
| $12C | ~32 | | | | | Channel B modulo high |
| $130 | Count32 | | | | | Channel B Count |
| $134 | ~32 | | | | |  |
| $140 | Address31..0 | | | | | Channel C address low |
| $144 | ~32 | | | | | Channel C address high |
| $148 | Offset31..0 | | | | | Channel C modulo low |
| $14C | ~32 | | | | | Channel C modulo high |
| $150 | Count32 | | | | | Channel C Count |
| $154 | ~32 | | | | |  |
| $160 | Address31..0 | | | | | Channel D address low |
| $164 | ~32 | | | | | Channel D address high |
| $168 | Offset31..0 | | | | | Channel D modulo low |
| $16C | ~32 | | | | | Channel D modulo high |
| $170 | Count32 | | | | | Channel D Count |
| $174 | ~32 | | | | |  |
| $178 | ~16 | | | Data16 | | Channel D Data |
| $17C | ~32 | | | | |  |
| $180 | BltSrcWid32 | | | | | Source width |
| $184 | ~32 | | | | |  |
| $188 | BltDstWid32 | | | | | Destination width |
| $18C | ~32 | | | | |  |
| $190 | ~16 | | | Op16 | | Blitter operation code |
| $194 | ~32 | | | | |  |
| $198 | ~2 | PLD6 | ~8 | BltCtrl16 | | Blitter Control |
| $19C | ~32 | | | | |  |

### Channel A Address

This pair of register sets the address of data source for channel A.

### Channel A Modulo

This pair of registers set the modulo amount for channel A. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

### Channel A Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

### Channel B Address

This pair of register sets the address of data source for channel B.

### Channel B Modulo

This pair of registers set the modulo amount for channel B. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

### Channel B Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

### Channel C Address

This pair of register sets the address of data source for channel C.

### Channel C Modulo

This pair of registers set the modulo amount for channel C. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

### Channel C Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

### Channel D Address

This pair of register sets the address of data destination for channel D.

### Channel D Modulo

This pair of registers set the modulo amount for channel D. The modulo amount is an amount added to the current working address once transfers have hit the destination width specification.

### Channel D Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

### Source Width

The source width specifies the number of horizontal pixels in the bitmap. It is used along with the modulo register to calculate the address of the bitmap data for a source.

### Destination Width

The destination width specifies the number of horizontal pixels in the bitmap. It is used along with the modulo register to calculate the address of the bitmap data for a source. As an example, the screen bitmap is 800 pixels wide, so the width value placed in the register would be 800.

### Blit Control

This register contains bits for control of the blit operation. Channels may be independently enabled or disabled. They may also be set to descending mode where the address decrements through memory instead of incrementing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bit | Default | Purpose |  |  |
| 0 | 0 | Channel A bitmap mode enable |  |  |
| 1 | 0 | Channel A DMA enable |  |  |
| 2 | 0 | Channel B bitmap mode enable |  |  |
| 3 | 0 | Channel B DMA enable |  |  |
| 4 | 0 | Channel C bitmap mode enable |  |  |
| 5 | 0 | Channel C DMA enable |  |  |
| 6 | 0 | reserved |  |  |
| 7 | 0 | reserved |  |  |
| 8 | 0 | Channel A descend mode |  |  |
| 9 | 0 | Channel B descend mode |  |  |
| 10 | 0 | Channel C descend mode |  |  |
| 11 | 0 | Channel D descend mode |  |  |
| 12 | 0 | reserved |  |  |
| 13 | 1 | Blit done indicator |  |  |
| 14 | 0 | Blit active indicator |  |  |
| 15 | 0 | Blit operation trigger |  |  |

### Blit Pipeline Depth ($198 bits 24 to 29)

This register controls the amount of pipelining present during the blit transfer. In some cases when data to be transferred is located nearby the pipeline depth may need to be reduced. The blitter normally makes use of queues of pixels to improve performance. Typically, the blitter works with 16 pixels in burst mode for any given channel. It will read 16 pixels for a channel before writing out the pixels to the destination. This may cause unexpected results in some circumstances.

The pipeline depth may also be reduced to give the main processor a greater share of memory access during the blit. It takes approximately n + 4 clock cycles for each burst read where n is the pipeline depth. If all four channels are active and the pipeline depth is 16, then this is approximately 80 clock cycles to perform one transfer. Note that while the blitter is performing a group transfer the main cpu will stall for the duration if it attempts to write the controller’s memory.

# Miscellaneous

`

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| $3E0 | ~8 | Strip Count8 | Z | ~7 | BPP3 | ~3 | Res2 |
| $3E4 | ~32 | | | | | | |

## Strip Count8

defunct

## Res2

defunct

## Z

Enable (1) or disable (0) 4-bit z-order mode. When 4-bit z-order mode is on the color selection is reduced to RGB444 but there are more “layers” to the display. When 4-bit z-order is off a full RGB555 color is possible.

Normal Pixel Layout in Memory

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | 14 10 | 9 5 | 4 0 |
| Z1 | Red5 | Green5 | Blue5 |

4-bit z-order Pixel Layout

|  |  |  |  |
| --- | --- | --- | --- |
| 15 12 | 11 8 | 7 4 | 3 0 |
| Z4 | Red4 | Green4 | Blue4 |

# Port Signals

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Width | I/O |  |
| rst\_i | 1 | i | This active high signal resets the core and WISHBONE bus interfaces |
| Slave Signals | | | |
| clk\_i | 1 | i | Clock signal for slave peripheral interface (typically the cpu clock) |
| cs\_i | 1 | i | circuit select |
| cyc\_i | 1 | i | cycle is valid |
| stb\_i | 1 | i | data transfer in progress |
| ack\_o | 1 | o | data transfer acknowledge |
| sel\_i | 8 | i | byte lane selects |
| we\_i | 1 | i | write enable to register set |
| adr\_i | 10 | i | addresses the registers of the core |
| dat\_i | 64 | i | data input for registers |
| dat\_o | 64 | o | data output of registers |
| Master Signals | | | |
| m\_clk\_i | 1 | i | clock signal for bus master interface (typically the memory clock) |
| m\_cyc\_o | 1 | o | cycle is valid |
| m\_stb\_o | 1 | o | data transfer is taking place |
| m\_ack\_i | 1 | i | data transfer acknowledge |
| m\_sel\_o | 16 | o | byte lane selects |
| m\_we\_o | 1 | o | write enable to memory |
| m\_adr\_o | 32 | o | Memory address for bitmap data read |
| m\_dat\_i | 128 | i | data input from bitmap memory |
| m\_dat\_o | 128 | o | this is data output to the memory |
| Video Port | | | |
| vclk | 1 | i | This is the video clock input (40 MHz) |