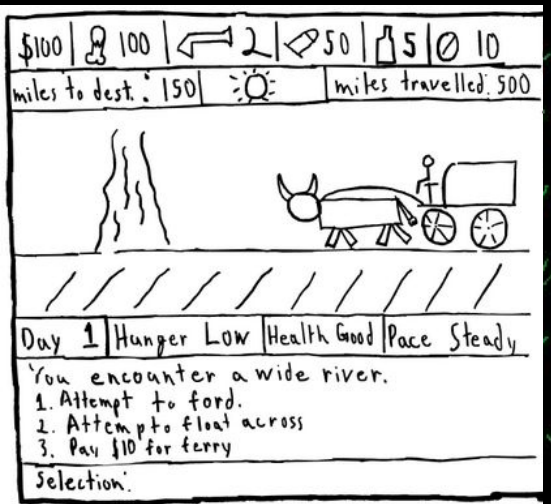
The Idaho Trail

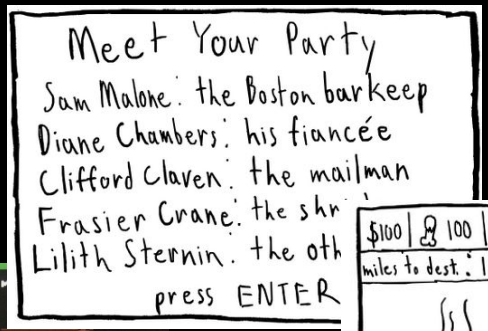
Almost the Oregon Trail

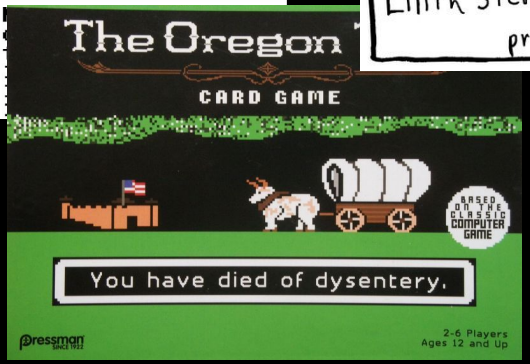
From Adam Auer, Sunny Li, and Xingcan "Ricardo" Chen

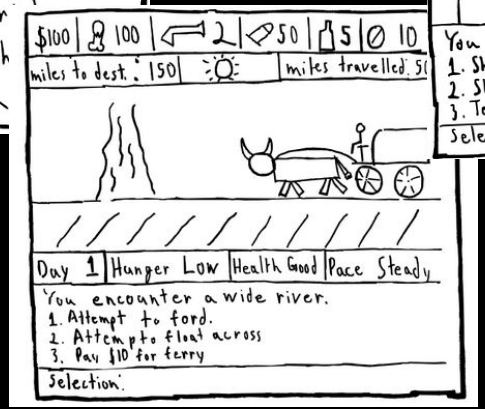


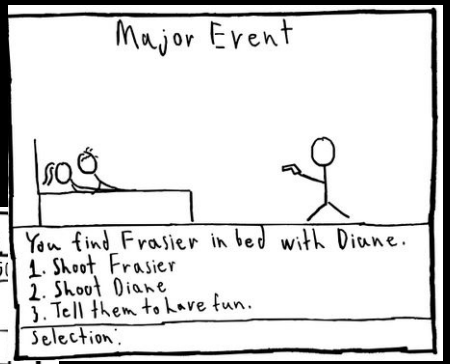
Motivation and Vision

- Passion for RPGs
- Many iterations of The Oregon Trail
- Many game mechanics





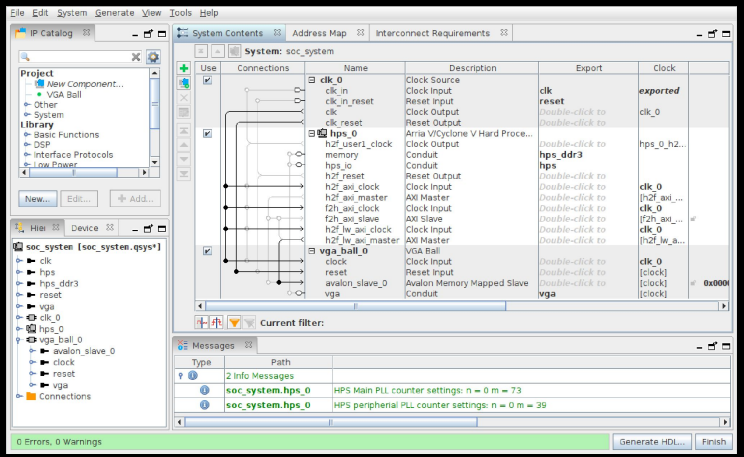


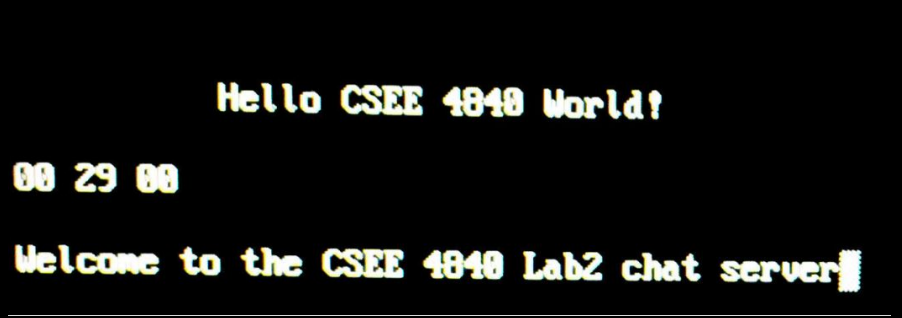


Challenges

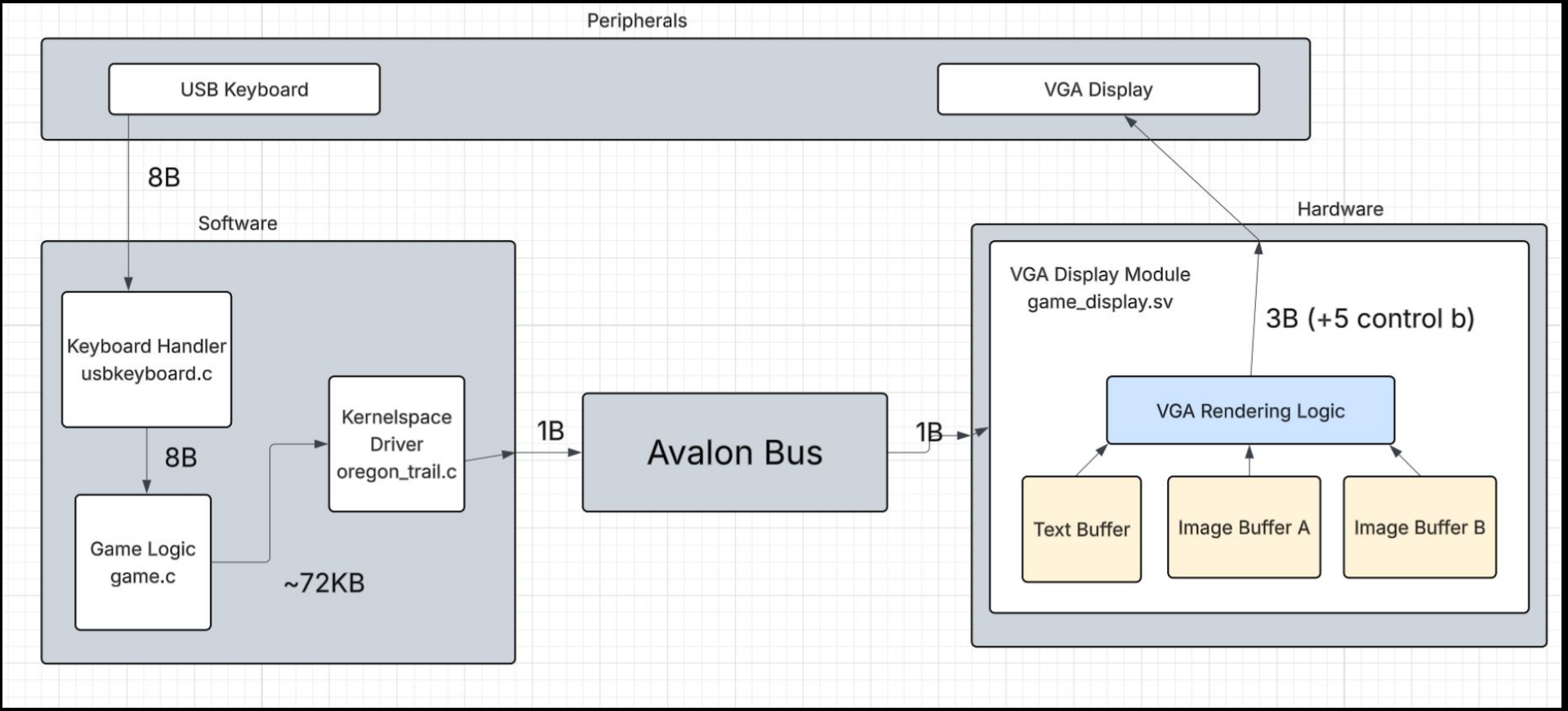
- Combine knowledge across labs
- Draw text
- Draw images
- Manage memory
- Leverage Intel Platform Designer

```
module collatz( input logic          clk,   // Clock
                input logic          go,    // Load value from n; start iterating
                input logic  [31:0]  n,     // Start value; only read when go = 1
                output logic [31:0]  dout,  // Iteration value: true after go = 1
                output logic         done); // True when dout reaches 1
```





Structure: Big Picture



Display: Physical Layout

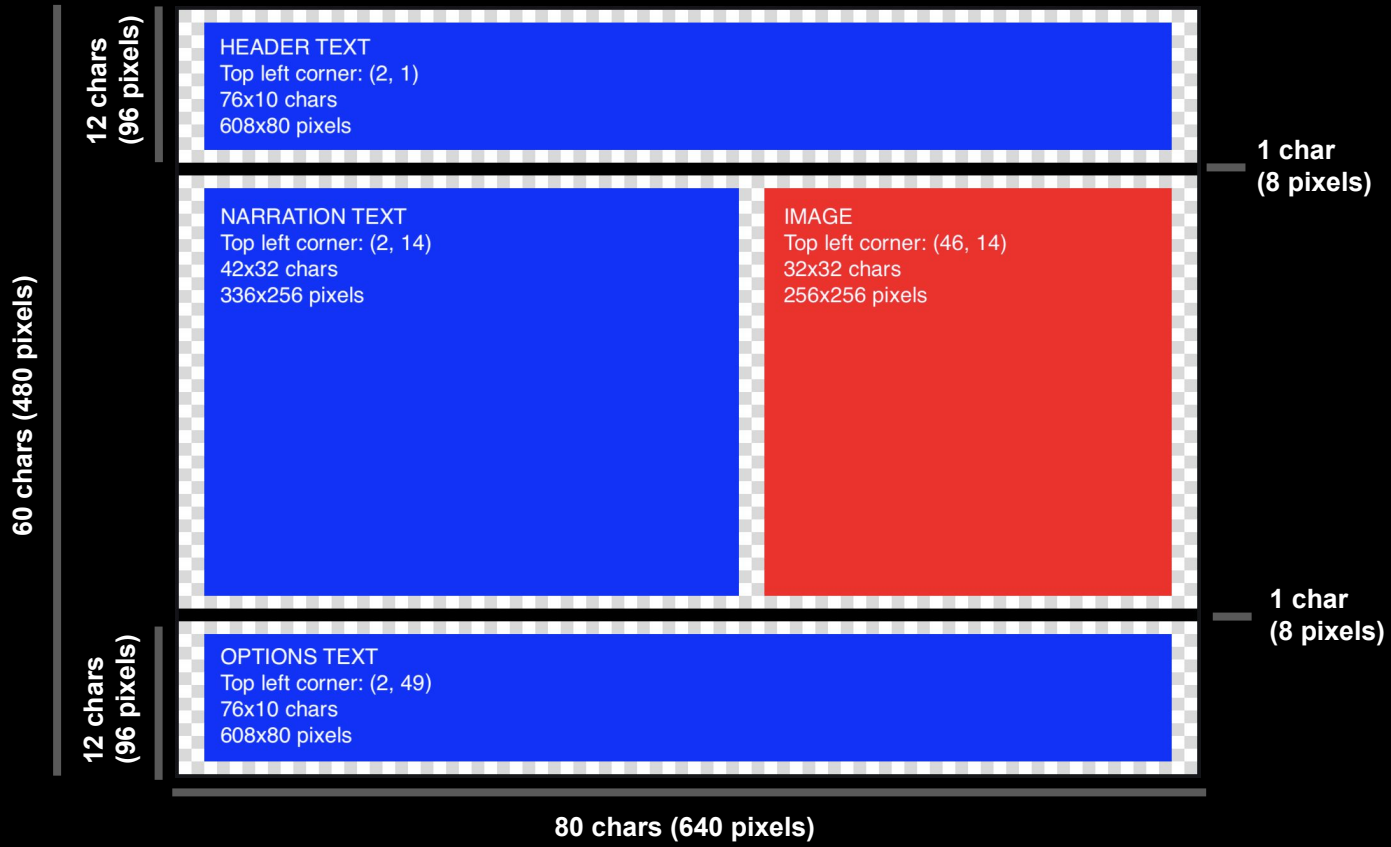
Blue: Text
Red: Image

1 char = 8x8 pixels

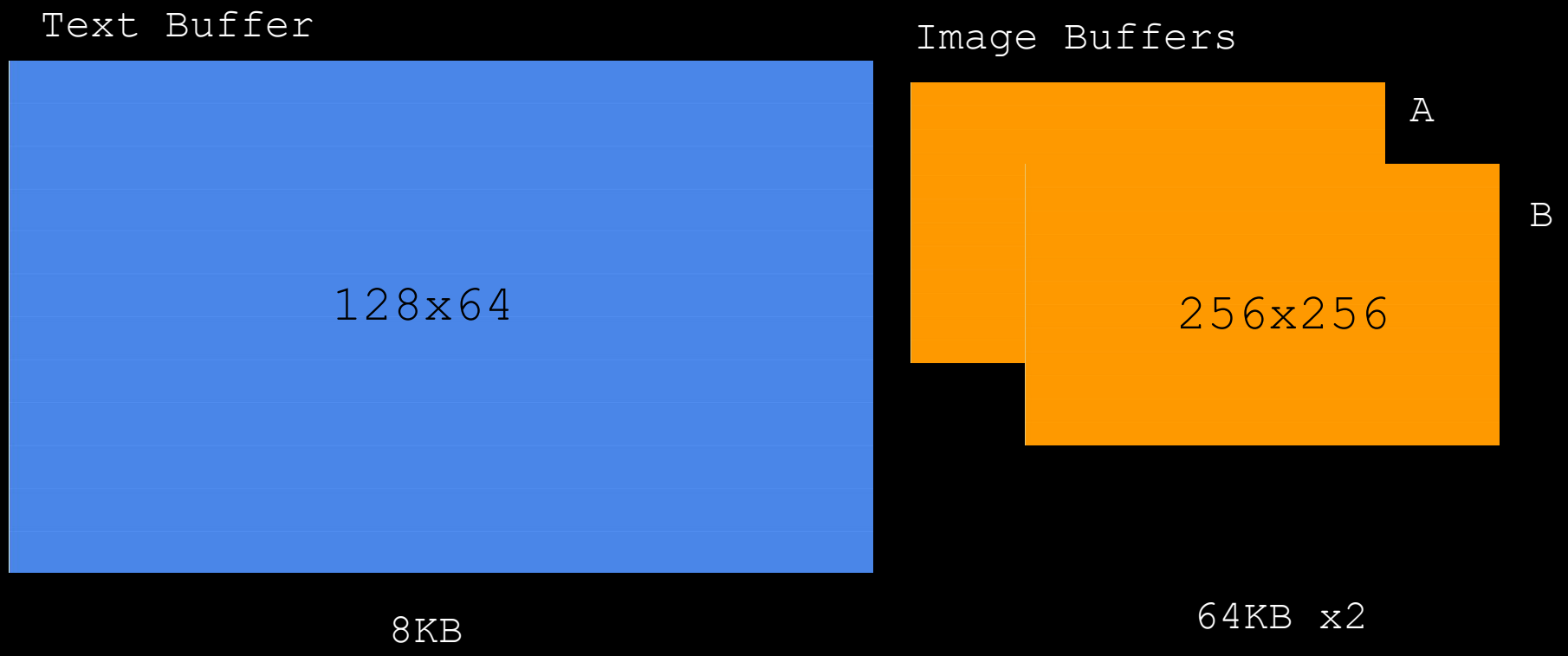
Unused pixels are padding

Horizontal Padding: **2 chars**

Vertical Padding: **1 char**



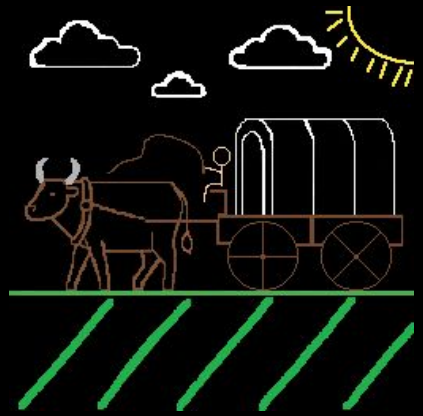
Display: Logical Layout



Gameplay Loop

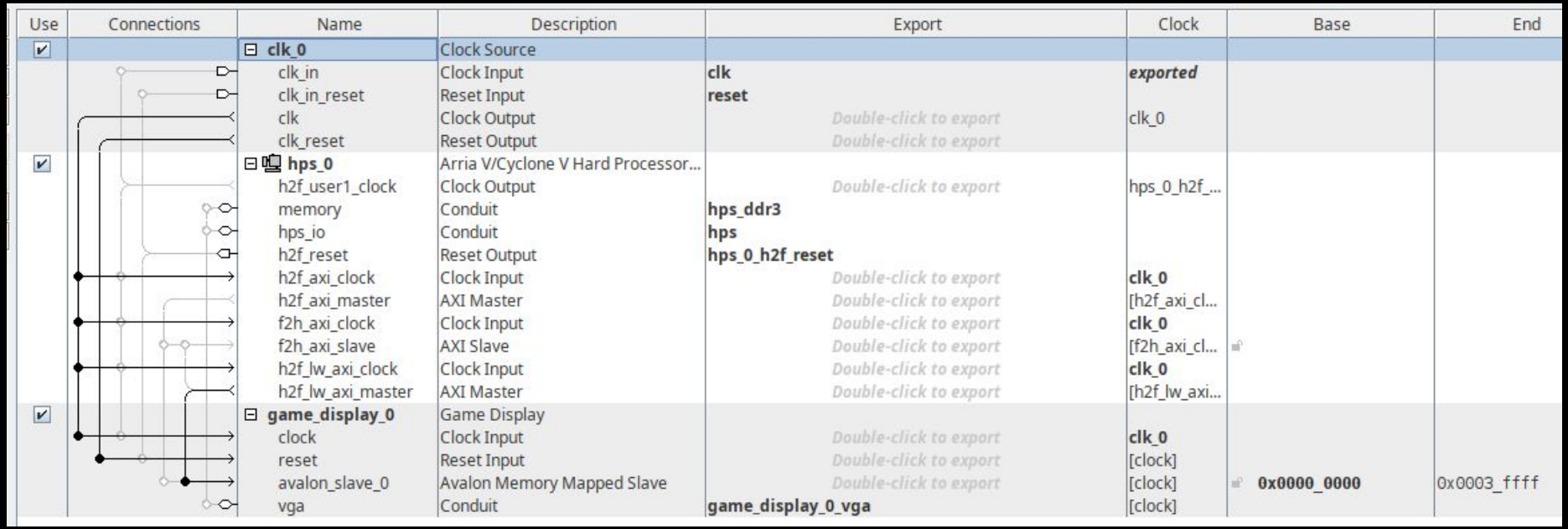
Wait for ‘1’, then start

1. Check death condition
   a. health >= 3 or
   b. food <= 0
2. Check victory condition: day = 5
3. Offer choice
   a. Continue on Trail -> advance day
   b. Shop -> purchase supplies
      i. Buy 1-9 lbs of food
      ii. Buy medicine (health improves by 1)
      iii. Leave store (go back to original page)
   c. Hunt -> chance for food
      i. No food
      ii. Gained (+5 food)
      iii. Injured during hunt (health damaged by 1)
   d. Rest -> gain 1 health
4. Check for victory/death
   a. health >= 3 || food <= 0 : death
   b. day >= 5: victory
5. At game end, loop back around to start screen



```
typedef struct {
  int health;
  int day;
  int money;
  int food;
  int scene;
  int screen_type;
} sw_state_t;
```

Connections

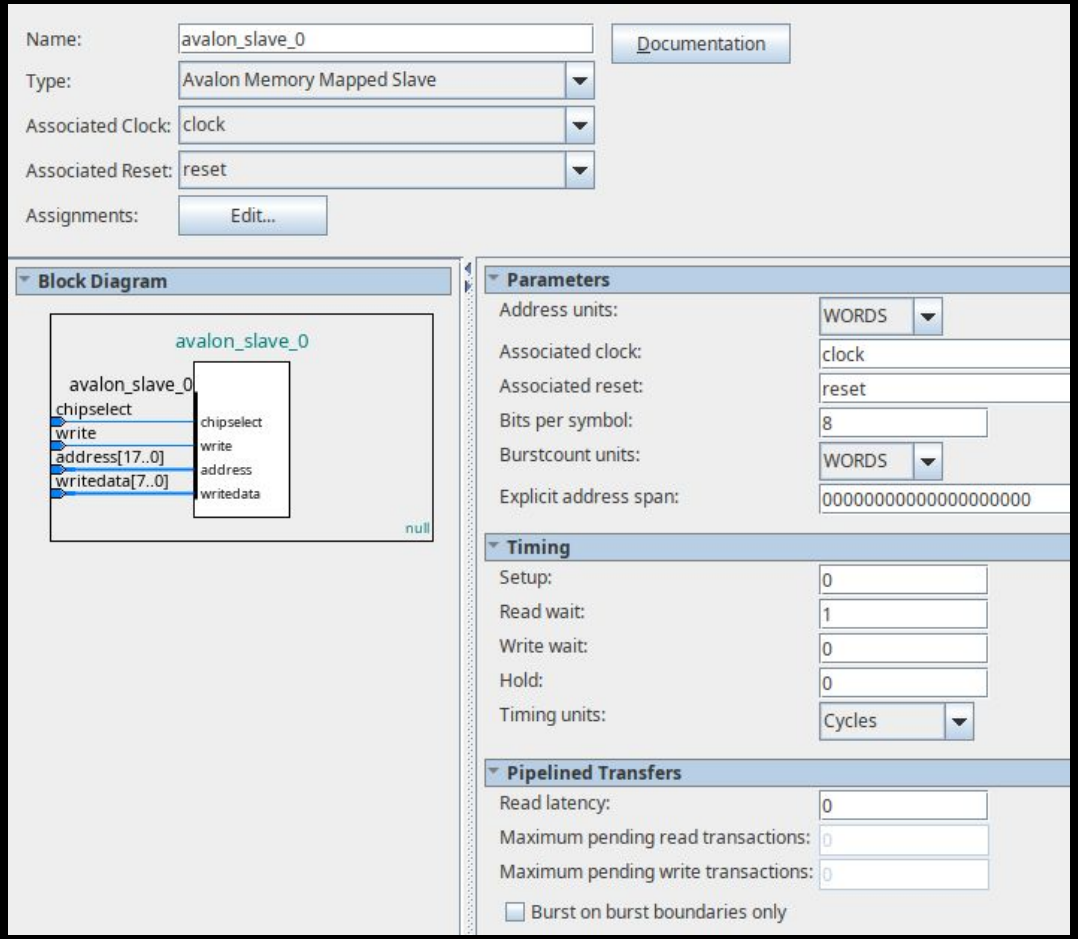


Use	Connections	Name	Description	Export	Clock	Base	End
☑		**clk_0**	Clock Source				
		clk_in	Clock Input	**clk**	***exported***		
		clk_in_reset	Reset Input	**reset**			
		clk	Clock Output	*Double-click to export*	clk_0		
		clk_reset	Reset Output	*Double-click to export*			
☑		**hps_0**	Arria V/Cyclone V Hard Processor...				
		h2f_user1_clock	Clock Output	*Double-click to export*	hps_0_h2f_...		
		memory	Conduit	**hps_ddr3**			
		hps_io	Conduit	**hps**			
		h2f_reset	Reset Output	**hps_0_h2f_reset**			
		h2f_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
		h2f_axi_master	AXI Master	*Double-click to export*	[h2f_axi_cl...		
		f2h_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
		f2h_axi_slave	AXI Slave	*Double-click to export*	[f2h_axi_cl...		
		h2f_lw_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
		h2f_lw_axi_master	AXI Master	*Double-click to export*	[h2f_lw_axi...		
☑		**game_display_0**	Game Display				
		clock	Clock Input	*Double-click to export*	**clk_0**		
		reset	Reset Input	*Double-click to export*	[clock]		
		avalon_slave_0	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	**0x0000_0000**	0x0003_ffff
		vga	Conduit	**game_display_0_vga**	[clock]		

```
assign VGA_CLK = clk;
```

VGA_CLK runs at 50 MHz

Interface


Name:
avalon_slave_0
Documentation
Type:
Avalon Memory Mapped Slave
Associated Clock:
clock
Associated Reset:
reset
Assignments:
Edit...
Block Diagram
avalon_slave_0
avalon_slave_0
chipselect
write
address[17..0]
writedata[7..0]
chipselect
write
address
writedata
null
Parameters
Address units:
WORDS
Associated clock:
clock
Associated reset:
reset
Bits per symbol:
8
Burstcount units:
WORDS
Explicit address span:
00000000000000000000
Timing
Setup:
0
Read wait:
1
Write wait:
0
Hold:
0
Timing units:
Cycles
Pipelined Transfers
Read latency:
0
Maximum pending read transactions:
0
Maximum pending write transactions:
0
Burst on burst boundaries only

Memory Inventory

	Size	Location	Desc.
Font	1 KB (128*8* 1)	BRAM	Loaded at synthesis time; used to draw characters
Image Buffer A	64 KB (256*256)	BRAM	Holds image for display or write
Image Buffer B	64 KB (256*256)	BRAM	Holds image for display or write
Text Buffer	8 KB (128 * 64)	BRAM	Holds text data; simultaneous read/write
Image Data (.hex)	192 KB	HPS Memory	Loaded on-demand by software

Total BRAM usage: 137 KB (Cyclone V max: 496 KB)

Register Map

Offset	Size	Name	Type	Access	Desc.
0x00000	1B	CONTROL	boolean	R/W	0 = blank screen 1 = screen on
0x00001	1B	SCREEN_TYPE	boolean	R/W	0 = right-side image 1 = center image
0x00002	1B	ACTIVE_BUF	boolean	R/W	0 = display from buf. A 1 = display from buf. B
0x0003-0x01FFF		*unmapped*			
0x02000-0x03FFF	8KB	TEXT_GRID	M10K BRAM	W	128x64 ASCII character grid
0x04000-0x0FFFF		*unmapped*			
0x10000-0x1FFFF	64KB	IMAGE_A_BUF	M10K BRAM	W	256x256 8bpp image grid
0x20000-0x2FFFF	64KB	IMAGE_B_BUF	M10K BRAM	W	256x256 8bpp image grid

The Core Structure

```
typedef struct
{
    __u8  control;       // 0 = Off, 1 = On
    __u8  screen_type;   // 0 = Gameplay, 1 = Fullscreen
    __u8  active_buffer; // 0 = Buffer A is active, 1 = Buffer B is active

    char  text_grid[TEXT_MEM_SIZE];
    __u8  image_data[IMG_MEM_SIZE];
} game_state_t;

typedef struct
{
  game_state_t state;
} game_arg_t;
```

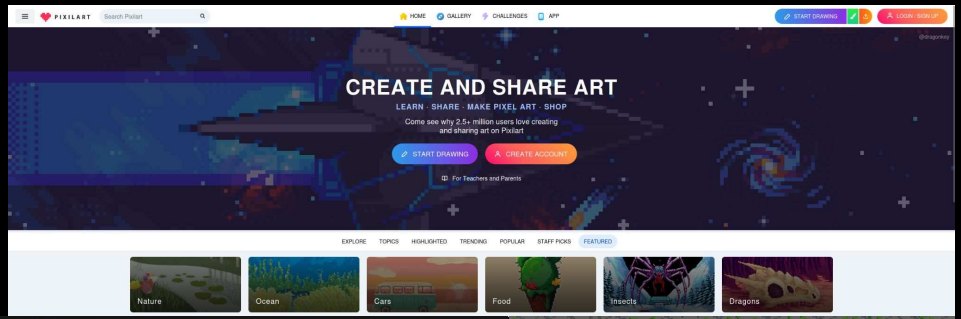
```
game_arg_t vla;
vla.state = hw_state;
ioctl(fd: oregon_trail_fd, request: GAME_WRITE_STATE, &vla);
```

Where the Rubber Meets the Road

```
static void write_game_state(game_state_t *state)
{
    void __iomem *text_ptr;
    void __iomem *img_ptr;
    int i;

    iowrite8(state->control, REG_CONTROL(dev.virtbase));
    iowrite8(state->screen_type, REG_SCREEN_TYPE(dev.virtbase));

    text_ptr = REG_TEXT(dev.virtbase);

    if (state->active_buffer == 0) {
        img_ptr = REG_IMG_A(dev.virtbase);
    } else {
        img_ptr = REG_IMG_B(dev.virtbase);
    }

    for (i = 0; i < TEXT_MEM_SIZE; i++) {
        iowrite8(state->text_grid[i], text_ptr + i);
    }

    for (i = 0; i < IMG_MEM_SIZE; i++) {
        iowrite8(state->image_data[i], img_ptr + i);
    }

    iowrite8(state->active_buffer, REG_ACTIVE_BUFFER(dev.virtbase));

    dev.state = *state;
}
```

Images: The Pipeline

1. Create Image as PNG in Pixilart
2. Convert PNG to .hex AoT using Python script
3. In software, load .hex file from disk and convert to binary pixel data
4. Load pixel data into BRAM buffer on the FPGA
5. Display logic determines final color values of every pixel, sets VGA signals




```
always_comb begin
    glyph_on = font_bits_s2[3'd7 - glyph_x_s2];
    img_pixel = active_buf_s2 ? img_b_s2 : img_a_s2;

    if (!display_enable_s2 || !visible_s2) begin
        final_pixel = 8'h00;
    end else if (screen_type_s2) begin
        final_pixel = in_img_s2 ? img_pixel : 8'h00;
    end else if ((text_char_s2 != 8'h20) && glyph_on) begin
        final_pixel = 8'hff;
    end else if ((text_char_s2 != 8'h20) && !glyph_on) begin
        final_pixel = 8'h00;
    end else if (in_img_s2) begin
        final_pixel = img_pixel;
    end else begin
        final_pixel = 8'h00;
    end
end
```

Credit to @DgGamez

Text: The Pipeline

1. Create .hex font file
   a. 128 glyphs
2. At synthesis time, parse and compile into .sof file
3. At FPGA configuration time, load binary font data into BRAM
4. Display logic determines final color values of every pixel, sets VGA signals

```
module font_sdp_rom (
    input  logic       clk,
    input  logic [9:0] raddr,
    output logic [7:0] rdata
);
    (* ramstyle = "M10K" *) logic [7:0] mem [0:1023];

    initial begin
        $readmemh("font.hex", mem);
    end

    always_ff @(posedge clk) begin
        rdata <= mem[raddr];
    end
endmodule
```

```
always_comb begin
    glyph_on = font_bits_s2[3'd7 - glyph_x_s2];
    img_pixel = active_buf_s2 ? img_b_s2 : img_a_s2;

    if (!display_enable_s2 || !visible_s2) begin
        final_pixel = 8'h00;
    end else if (screen_type_s2) begin
        final_pixel = in_img_s2 ? img_pixel : 8'h00;
    end else if ((text_char_s2 != 8'h20) && glyph_on) begin
        final_pixel = 8'hff;
    end else if ((text_char_s2 != 8'h20) && !glyph_on) begin
        final_pixel = 8'h00;
    end else if (in_img_s2) begin
        final_pixel = img_pixel;
    end else begin
        final_pixel = 8'h00;
    end
end
```

```
00
00
00
00
00
00
00
00
7E
81
A5
81
BD
99
81
7E
```

DEMONSTRATION

PRESS 1 TO START