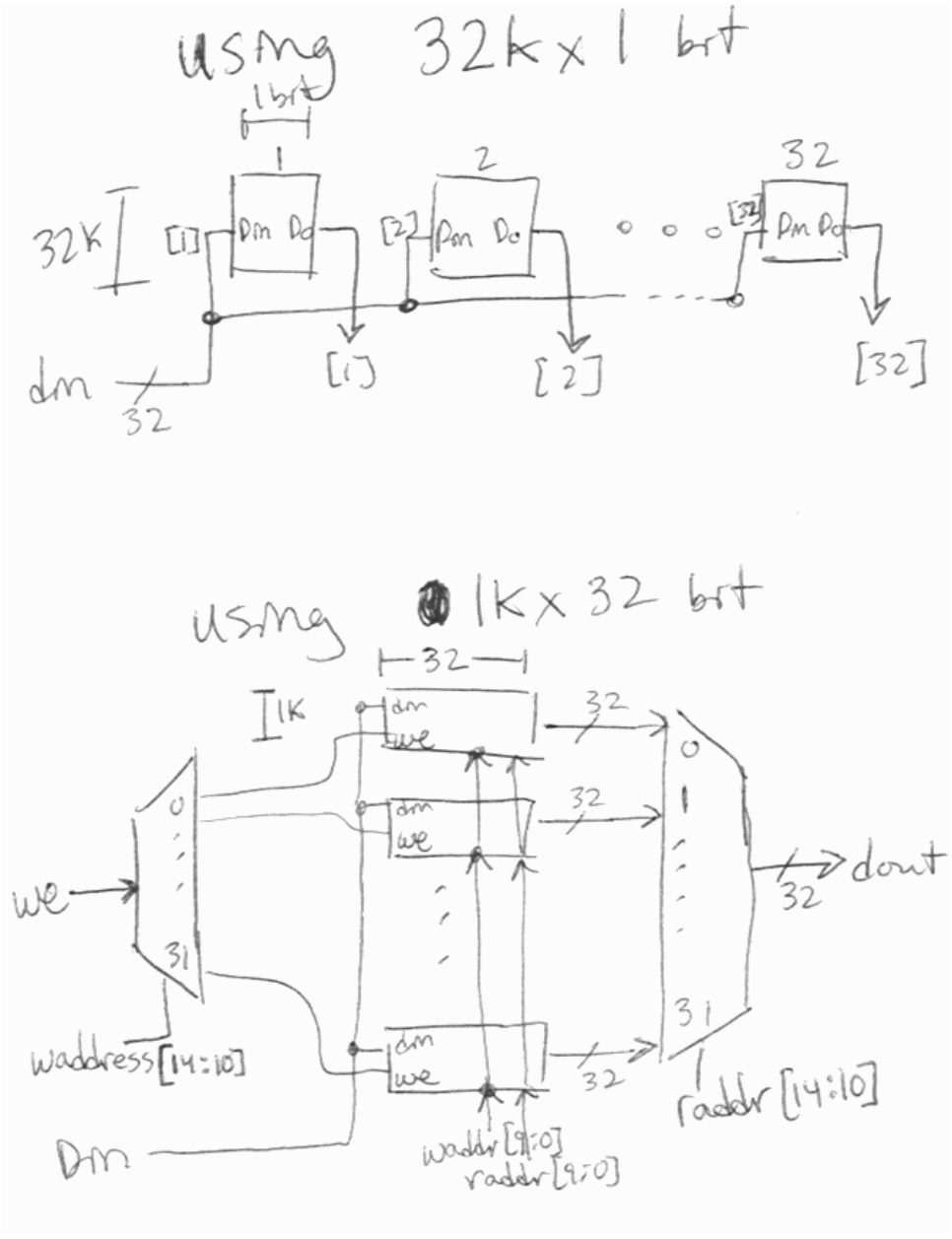
University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS150, Spring 2010

Homework Assignment 7 Solutions: Memories and Video

1. Your task is to implement a 32K x 32 memory. Assume you are given 32Kbit configurable memory blocks such as a block RAM in a FPGA. Each block can be configured as a 32K x 1bit memory or a 1K x 32bit memory. Which option would you choose as a building block for your 32K x 32bit memory and why?

   Drawing the 32Kx32 memory using each of the building blocks, it seems clear that 32Kx1 building block is the best choice. It uses the least extra logic and therefore could have better performance and lower area cost.
   Using the 32Kx1 memories, the larger memory can be constructed by simply using the one bit from each block as one of the 32 bits of data. On writes, the address goes to all blocks, write-enable is asserted for all, and one of the 32 bits goes to each block.
   Using the 1Kx32 memories, each data word is stored in one of the 32 blocks. So, for reads, part of the address chooses the word inside the blocks and the rest of the address chooses between the data from the blocks. For writes the dataIn goes to all blocks but only the appropriate write-enable is asserted, determined by a decoder. So, the 1Kx32 implementation requires an extra 32:1 decoder and a 32-bit wide 32:1 mux; this is a lot of extra hardware and if both memory blocks are assumed the same, then performance is also worse because of the extra delay through the decoder on writes and mux on reads.



2. (a) Write a parameterized memory in Verilog.

```
module RAM(Clock, WAddress, RAddress, WE, DataIn, DataOut);

parameter Width = 16,
          Depth = 1024*4,
          SynchronousRead = 0;

// assume a macro `log2 is defined
localparam AWidth = `log2(Depth);
```

```
input Clock;
input [AWidth-1:0] WAddress, RAddress;
input WE;
input [Width-1:0] DataIn;
output [Width-1:0] DataOut;

reg [Width-1:0] mem [Depth-1:0];

reg [AWidth-1:0] ReadAddress_p;

always @ (posedge Clock) begin
    if (WE) begin
        mem[WAddress] <= DataIn;
    end
end

generate
if (SynchronousRead) begin
    always @ (posedge Clock) begin
        ReadAddress_p <= RAddress;
    end
    assign DataOut = mem[ReadAddress_p];
end else begin
    assign DataOut = mem[RAddress];
end
endgenerate

endmodule
```

(b) Create a Xilinx ISE project and add your RAM module. Set the parameters to implement each of the following memories. For each, record the type of resource used, the percentage of this type of resource that is used up, and the memory configuration used. This information is near the bottom of the synthesis report.

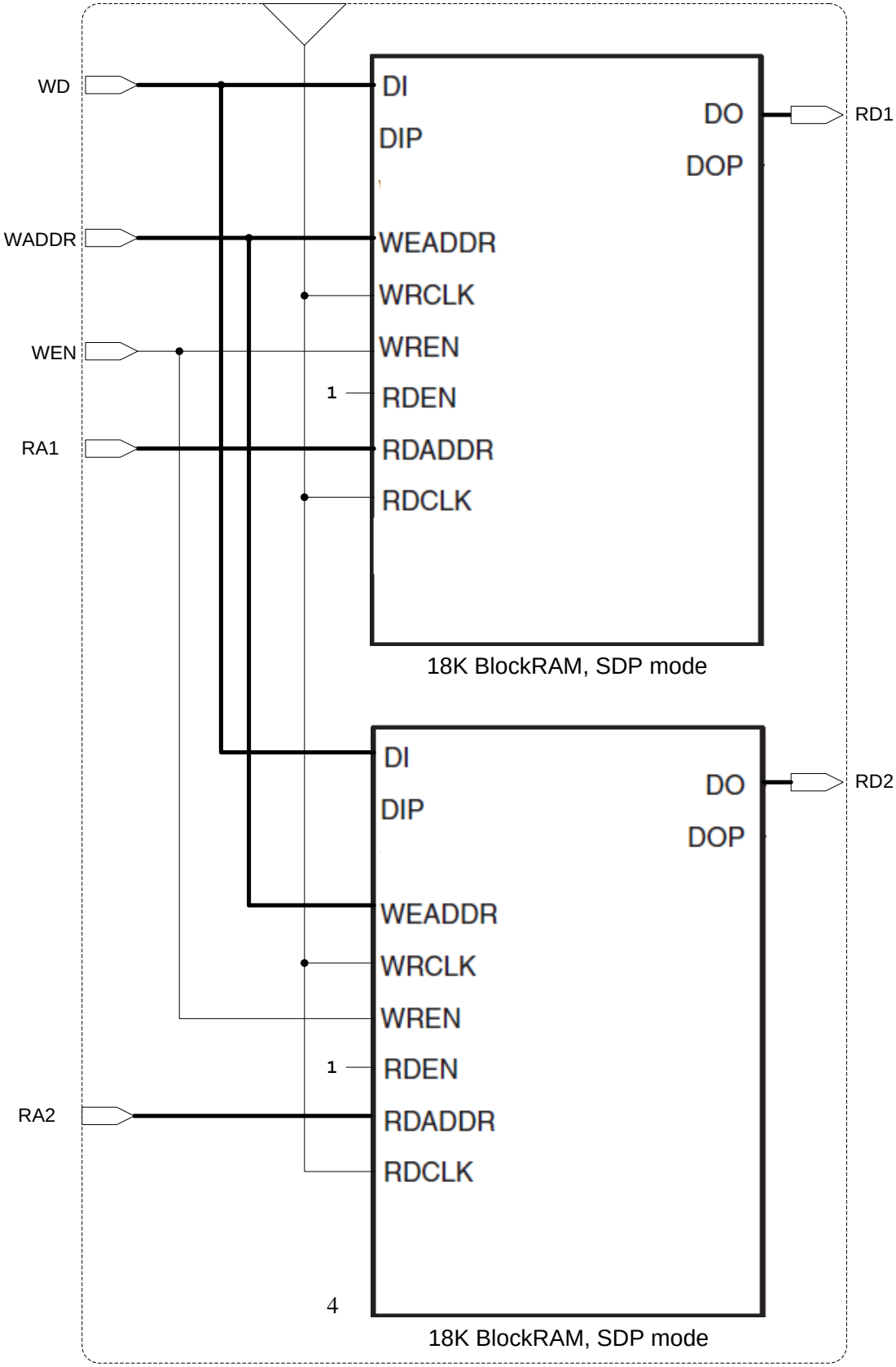
i. depth=32, width=1, SynchronousRead=0.
Dual Port Rams (RAM32X1D): 1
LUTs: 2 (0%)

ii. depth=4096, width=16, SynchronousRead=0.
Dual Port Rams (RAM128X1D): 512
LUTs: 2240 (3%)

iii. depth=32*1024, width=32, SynchronousRead=1.
Block Rams : 32 of 148 (21%)

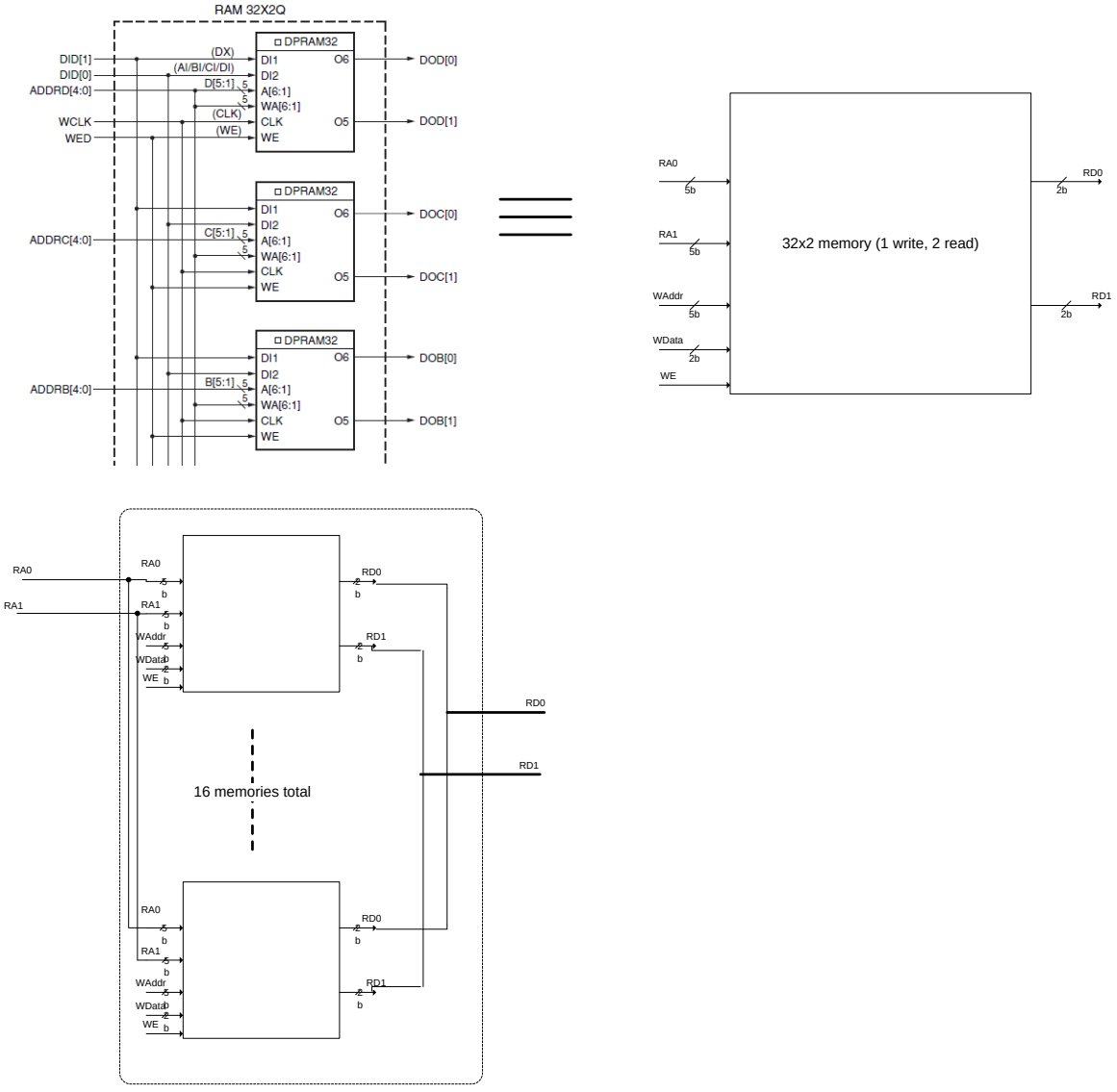
3. We want to build a 32 x 32bit register file, with 2 read ports and 1 write port, on an FPGA. Refer to Virtex-5 FPGA User Guide (ug190.pdf) on the Documents page for this problem.

(a) Build the register file from Block RAM, using as few resources as possible. Description of Virtex-5 Block RAM can be found starting at pg.109 of the User Guide. Draw your design and give a brief description, including what configuration is used.


WD
WADDR
WEN
RA1
RA2
DI
DIP
DO
DOP
WEADDR
WRCLK
WREN
1
RDEN
RDADDR
RDCLK
RD1
RD2
18K BlockRAM, SDP mode
18K BlockRAM, SDP mode

To get a data width of 32, we can use a 18Kb BlockRAM in simple dual-port mode, which allows up to 36 bit data width and independent read and write. We need 2 reads and 1 write on every cycle, so we need to replicate the BlockRAM memory to gain an extra read port. The write port for both RAMs will use the same signals. Using BlockRAMs for a 32x32 regfile wastes most rows in the memory.

(b) Build the register file from Distributed RAM (LUT RAM), using as few resources as possible. The basic primitives are found on pg.207 of the User Guide. There may also be useful information within *Chapter 5: Configurable Logic Blocks (CLBs)*. Draw your design and give a brief description, including what configuration is used.



To get a data width of 32 and multiple ports, we choose the LUTRAM 32x2 Quad port

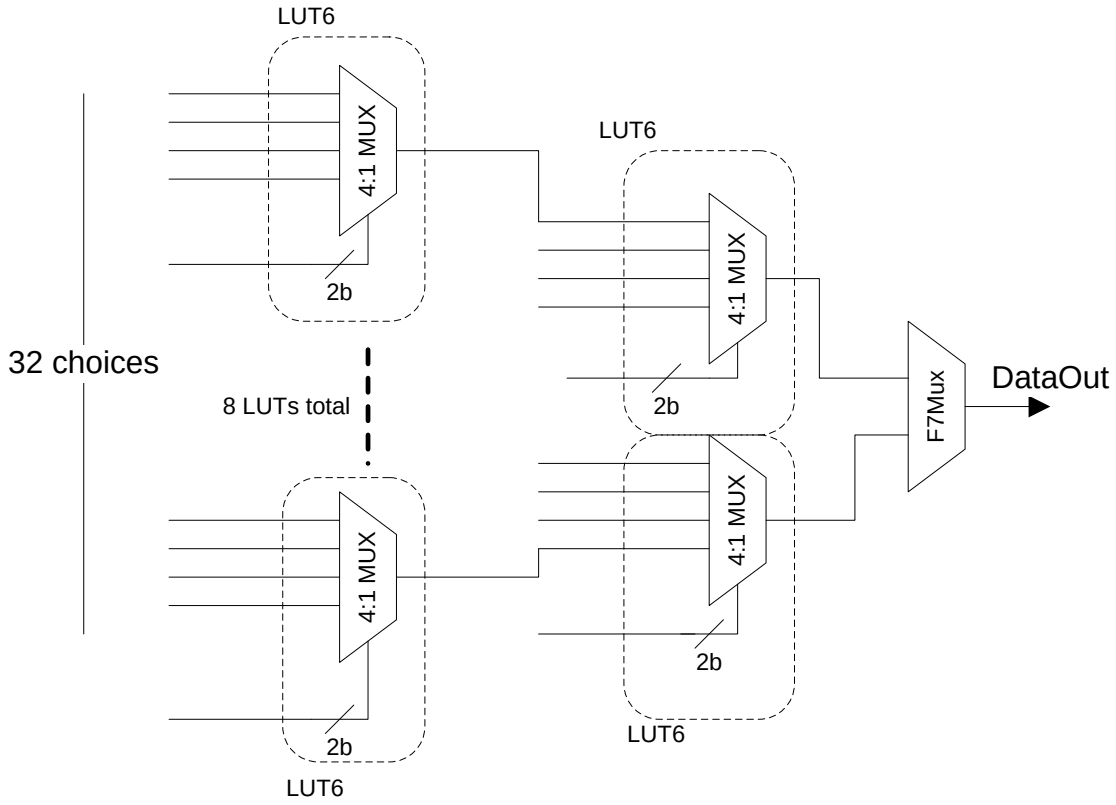
configuration. One port is the SLICEM write signals, which go to all 4 LUT6's within the SLICEM. Two LUTs, with their independent read addresses can provide the two read ports. The fourth LUT6 is unused (technically the first one is too, but we need to use its address signals for write address). Inside each LUT6, the two LUT5's that comprise it each provide a bit of data, and we know that a LUT5 has 32 bits of state, thus 32x2 memory. We can compose 16 of these 32x2 1W2R memories in parallel to get the 32-bit data width.

(c) Assume the rough area costs below:

Component	Area (μm^2)
LUT5	850
LUT6	1700
BRAM36	370000
BRAM18	185000
FlipFlop	100

i. Approximately how much area does it take for 36Kbits of storage using:
   A. Block RAM? 36Kbit*(1 BRAM36/36Kbit) = $370000\mu m^2$
   B. LUT RAM? 36Kbit*(1 LUT6/64bit) = $979200\mu m^2$
   C. FlipFlops? 36Kbit*(1 FF/1 bit) = $3686400\mu m^2$

ii. How much area does each of your register file implementations require?
   A. Block RAM. 2*BRAM18 = $370000\mu m^2$
   B. LUT RAM. 3 LUT6 per quad port 32x2 memory (when excluding unused fourth port). Need 16 of these in parallel to have 32-bit wide data. 16*3*LUT6 = $81600\mu m^2$.
   C. FlipFlops (estimate based on number of flipflops, as well LUTs for surrounding logic).
   Each bit of 5:32 decoder (with WE input) can fit in a LUT6, so 32 LUT6 for decoder.
   32*32=1024 FlipFlops for the memory.
   10 LUT6 for each bit of the 32:1 Mux. For 32 data width, we use 32 copies of this, so: 10*32=320 LUT6. We have two read ports, so 2 muxes: 2*320 = 640 LUT6. This is one possible mapping of the 32:1 Mux to LUTs. Below is a schematic of 1 copy (1-bit wide 32:1 Mux).



$(32 + 640) * LUT6 + 1024 * FF = 1244800\mu m^2$

A more efficient implementation of the 32:1 mux is to use the F7Muxes and F8Mux to create a 16:1 Mux. Two of these 16:1 Muxes can be combined with a 2:1 Mux (using another LUT). This adds up to 9 LUT6's per 1-bit 32:1 Mux.

$(32 + 576) * LUT6 = 1136000\mu m^2$

From this exercise we first see that BlockRAM is the most dense memory (approximately 10x more dense than flipflops) and desirable for larger memories. We also saw an example of how different memory types compare for a given application. For a 32x32bit register file, BlockRAM wastes resources because we need two to have enough ports, but each BlockRAM already has far more space than needed. LUTRAM takes only $81600\mu m^2$ despite needing a replica of the data for each port. Flipflop memory does not need to be replicated for 1 write port and any number of read ports, but since it is less dense, the flipflops themselves are more area (about $20000\mu m^2$ more) than the LUTs, and also need surrounding logic: decoder for writes and muxes for reads.

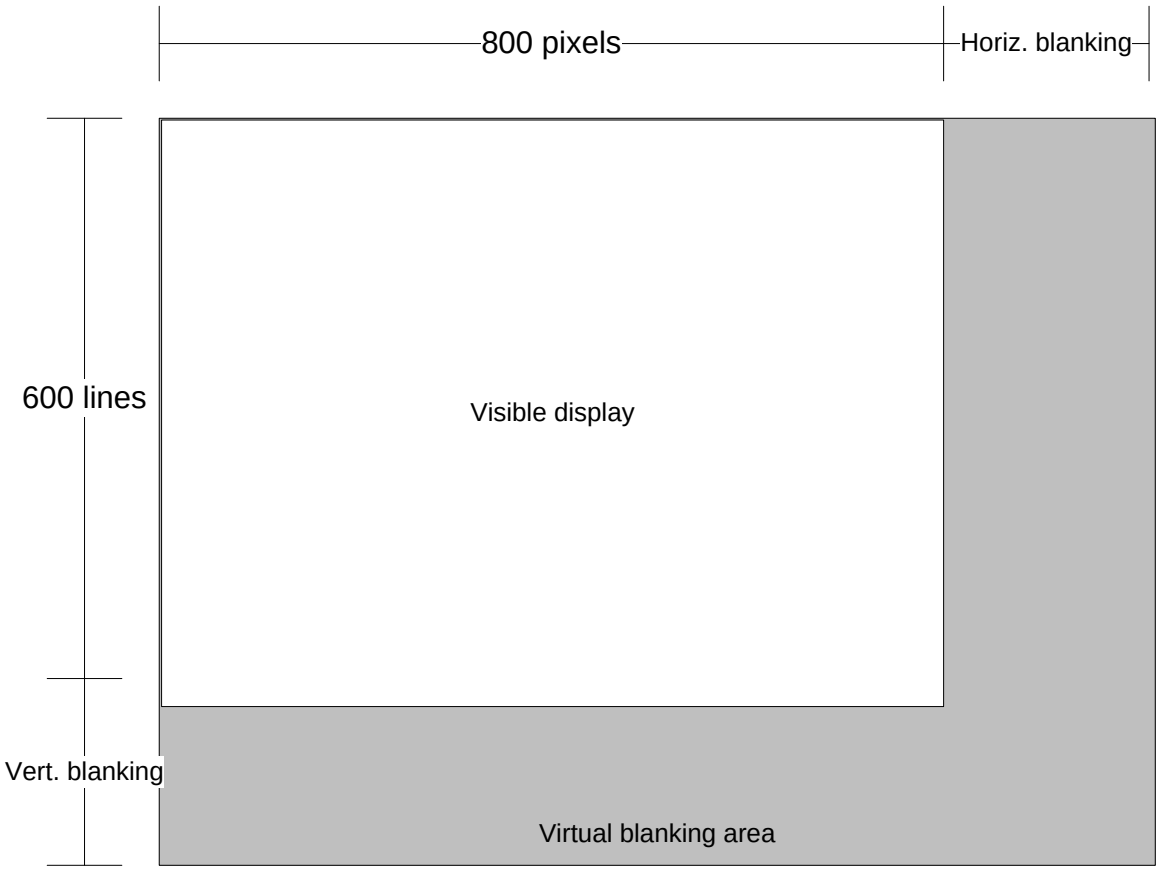
4. We are using a 800x600 display at 75Hz, meaning 75 frames are sent to the display every second.

   (a) What is the pixel rate?
   800 pixels/line * 600 lines/frame * 75 frames/sec = $36 * 10^6$ pixels/sec

   (b) For 24-bit color, what is the data rate?
   $36 * 10^6$ pixels/sec * 24 bit/pixel = $864 * 10^6$ bit/sec

(c) What is the pixel frequency (clock used by display) for sending pixels to our 800x600 @ 75Hz display?
let h=lines/frame, w=pixels/line, hb=horizontal blanking interval, vb=vertical blanking interval, fr=frame rate, pf=pixel frequency (the unknown)
$h * (w/pf + hb) + vb = 1/fr$
$600 lines/frame * ((800 pixels/line)/pf + 5.17171717 \mu s) + 0.5333333 ms/frame = 1/75 sec/frame$
$pf = 800/((1/75 - 0.5333333 * 10^{-3})/600 - 5.17171717 * 10^{-6})$
$pf = 49.5 * 10^6 pixels/sec$

(d) What is the horizontal blanking interval in terms of number of pixels?
One way to think of blanking intervals is as additional display area offscreen as shown below.



We know the pixel frequency from (c) and just multiply this by the horizontal blanking interval to get how many pixels.
$pf * hb$
$49.5 * 10^6 pixel/sec * 5.17171717 * 10^{-6} s = 256 pixels$
What is the vertical blanking interval in terms of number of lines?
Part (c) mentioned that the vertical blanking interval is like writing extra offscreen lines at

the end of each frame. We know the time per line (including blanking interval) by plugging in the pixel frequency. Divide the vertical blanking interval by time per line to get number of lines.
$vb/(w/pf + hb)$
$0.5333333 * 10^{-3}/((800 pixels/line)/(49.5 * 10^{6} pixels/sec) + 5.17171717 * 10^{-6})$
$25 lines$

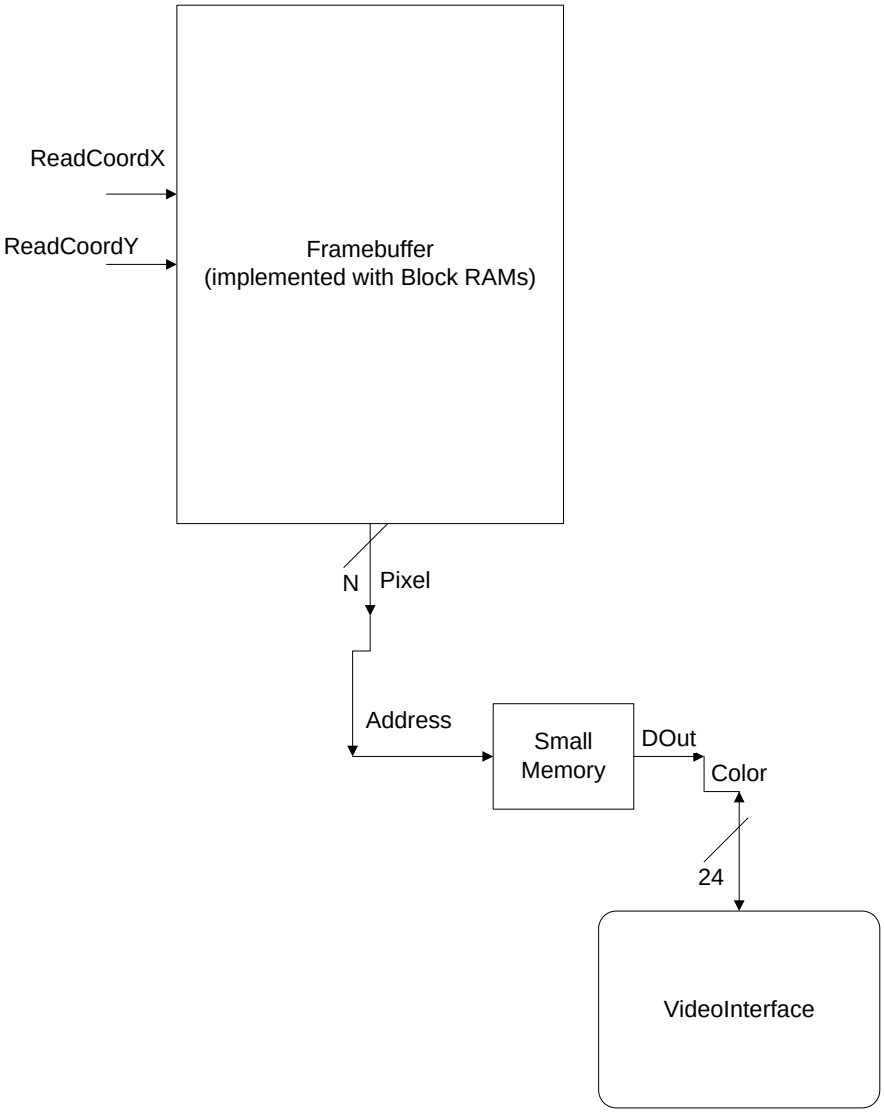
The spec for the display used in this problem can be found at http://tinyvga.com/vga-timing.

5. Suppose we implement our 800x600 pixel framebuffer using the Block RAM resources on our XC5VLX110T FPGA. You can look in "Virtex-5 Family Overview" on the Documents page of the website for the resources available on different Virtex-5 FPGA models. Suppose half the Block RAMs on the fpga are being used for our instruction and data memories already, leaving half for the framebuffer.

   (a) How many bits are available for each pixel?
   The Family Overview says there are 5,328Kib of BlockRAM memory on the FPGA.
   $5328 * 1024/(800 * 600) = 11.3664 bits/pixel$
   The problem says half of the BlockRAMs are available so 5.6832 bits/pixel. There is not a 5-bit data width configuration for BlockRAMs, so if we were implementing this framebuffer, we might store each pixel across 5 BlockRAMs, using the 32Kx1 configuration (although the addressing may be easier if we just used 4-bit pixels and the 8Kx4 configuration).

   (b) How many distinct colors can be represented? (Let this number be hereafter called N)
   5 bits per pixel to encode the color. $2^5 = 32 = N$.

   (c) Recall that our video interface supports 24-bit color. How can you use a relatively small memory to allow any N of the 2^{24} displayable colors to be used at a given time? Draw a high level block diagram of the framebuffer, video interface, and the extra memory block implementing this functionality. Specify the dimensions of the small memory.



The small memory is 32x24 bit. It has 32 24-bit color entries. This table turns a 5-bit color code into a 24-bit color. Thus, at one time we can display up to 32 out of the 2^{24} displayable colors.

(d) The CPU should be able to write to this small memory (sometimes called a "Colormap") to change the colors being used and even do short animations efficiently. Suppose the Colormap is in the CPU memory map at addresses 0xA0000000 - 0xA000007C (remember memory is byte addressed but we only use word loads and stores so the 2 least significant bits of the address are always zero). Suppose initially both the framebuffer and Colormap are filled with 0's in every memory location. Fill in the code in main() that does an animation of a green vertical bar moving across the screen, with a black background. Make

the animation occur at roughly 25 frames per second. Notice helpful functions sleep() and frameBufferAddress() are provided.

```
void sleep(long delay) {
    // ...makes CPU wait for 'delay' milliseconds...
}

int frameBufferAddress(int x, int y) {
    // ...returns the framebuffer memory address
    // corresponding to pixel (x,y)...
}

int main() {
    int BLACK = 0x000000;
    int GREEN = 0x00FF00;
    int colorMapBaseAddress = 0xA0000000;

    // write bars to the framebuffer, with color values 1 thru N-1.
    for (int x=1; x<N; x++) {
        for (int y=0; y<600; y++) {
            *(frameBufferAddress(x, y)) = x;  // write x to pixel (x,y)
        }
    }

    // Animation - animate a GREEN vertical bar moving across the screen
    // from x=1 to x=N-1.
    for (int x=1; x<N; x++) {
        int olda = colorMapBaseAddress + 4*(x-1);
        int newa = colorMapBaseAddress + 4*x;
        *olda = BLACK;
        *newa = GREEN;
        sleep(40);
    }
}
```