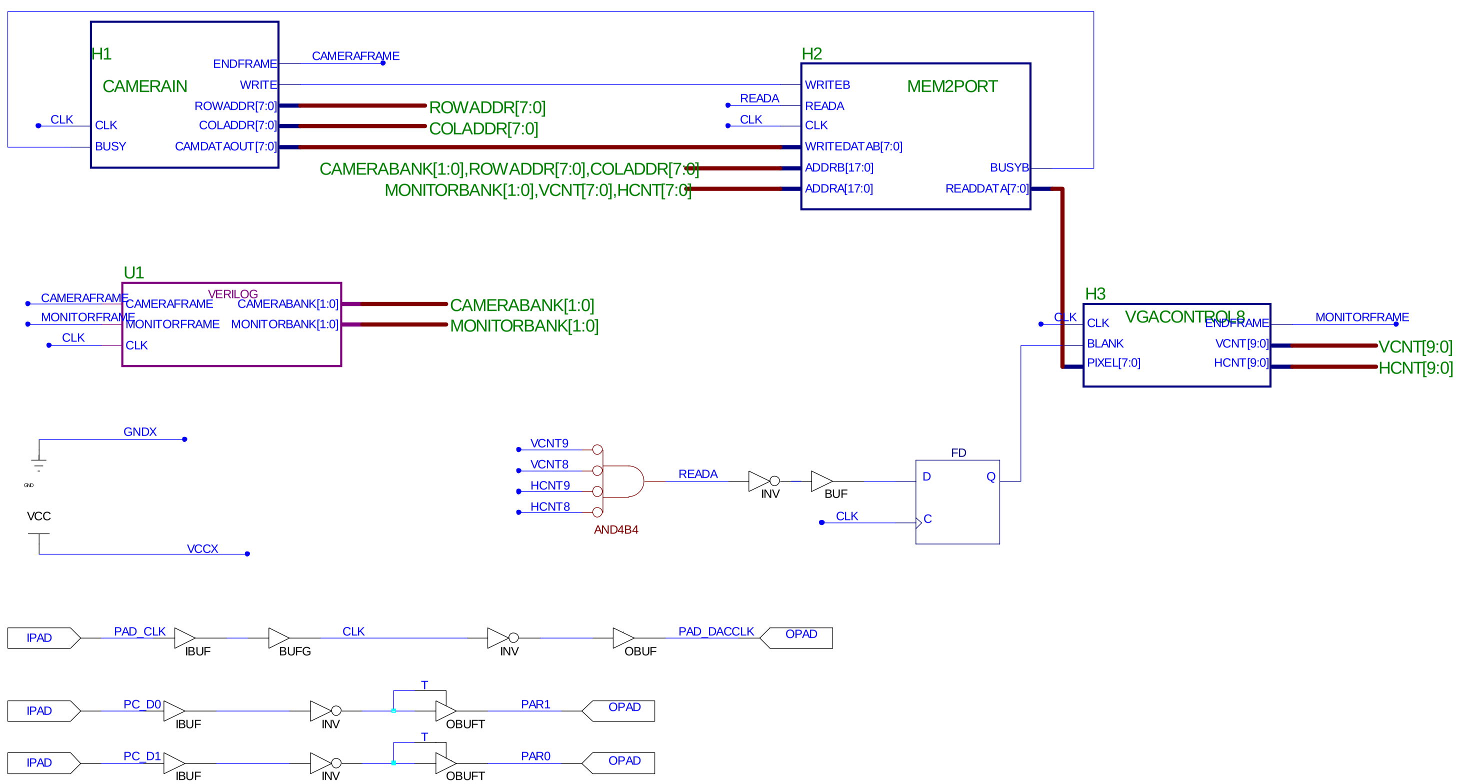
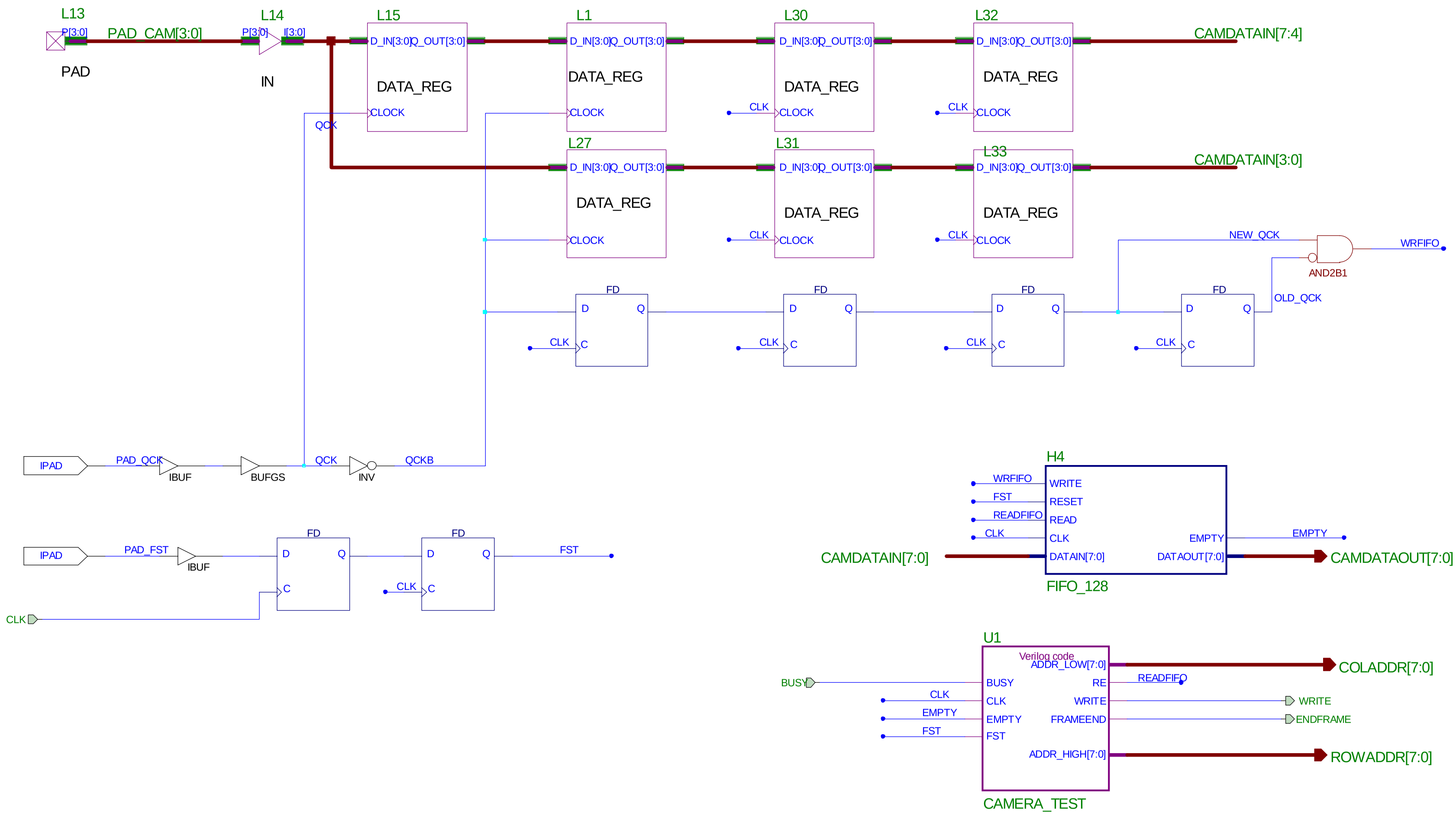
TOP

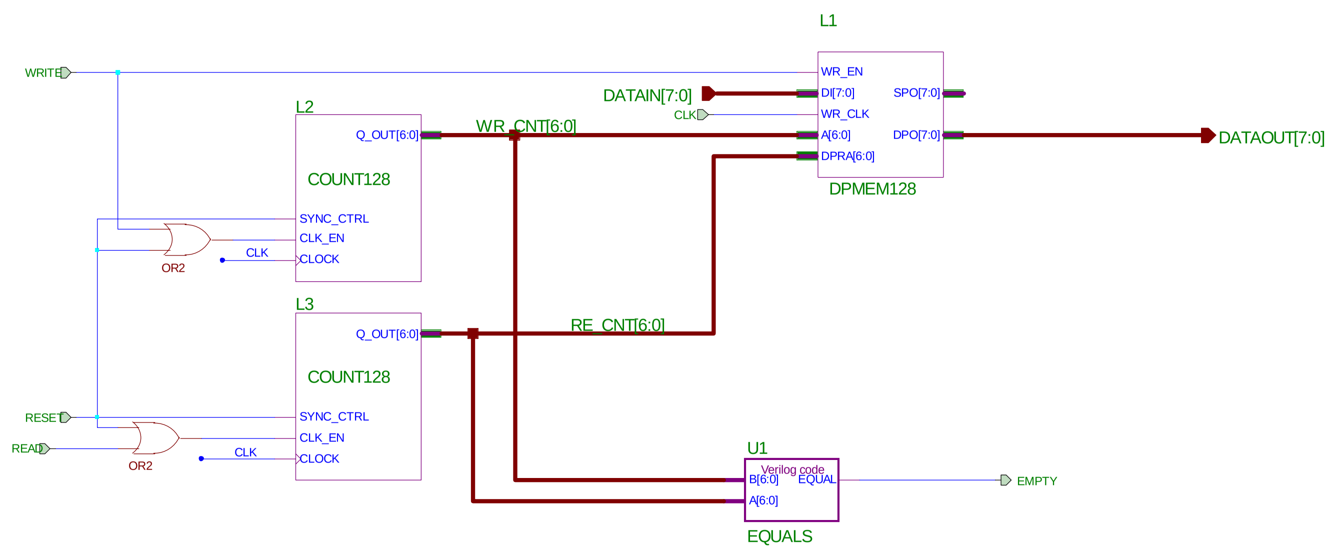


Xilinx Corporation	Project: VGACAM
2100 Logic Drive	Macro: TOP
San Jose, CA 95124	Date: 5/14/99
Date Last Modified: 1/17/0	



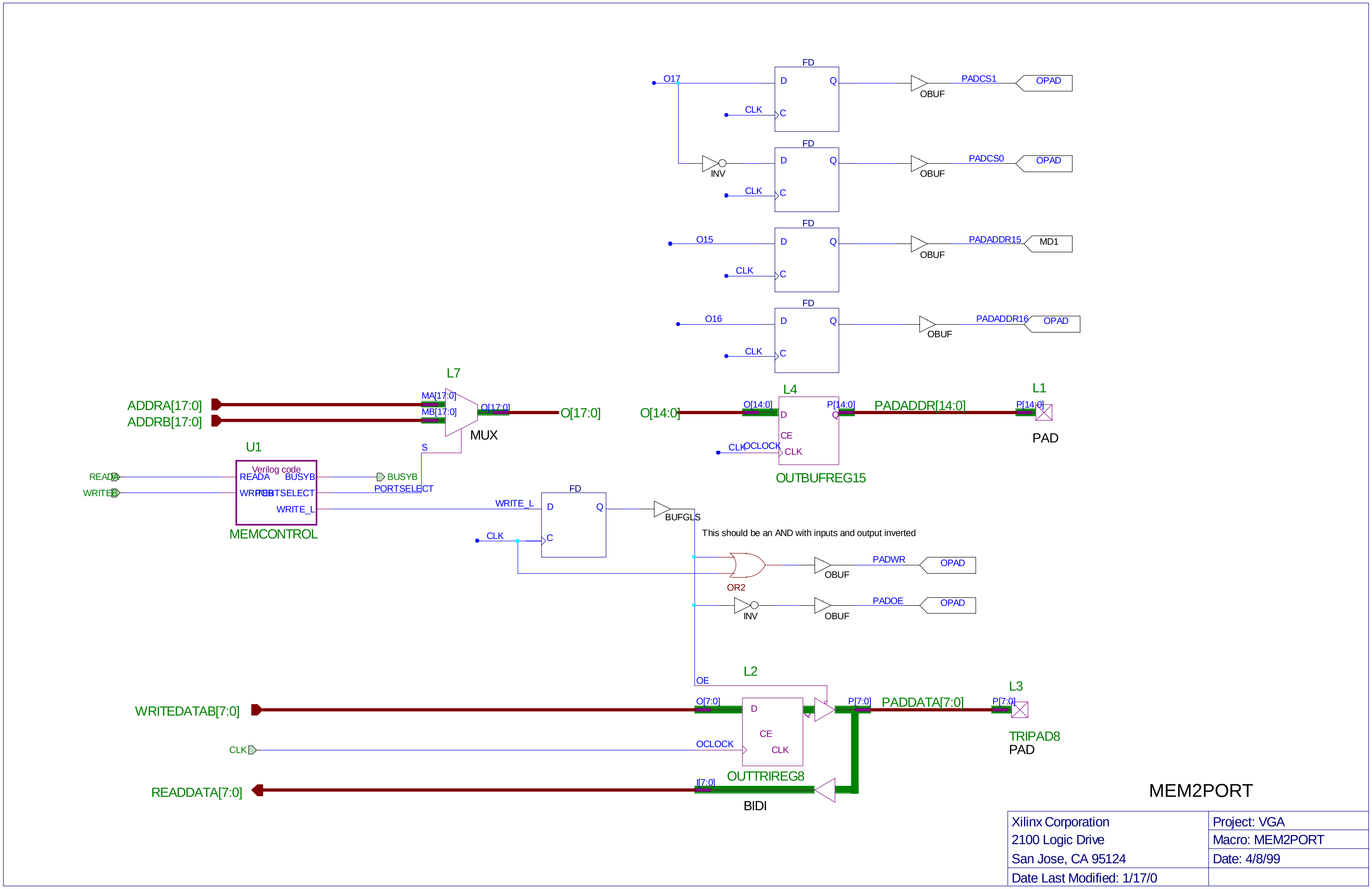
CAMERAIN

Xilinx Corporation	Project: VGACAM
2100 Logic Drive	Macro: CAMERAIN
San Jose, CA 95124	Date: 5/14/99
Date Last Modified: 1/17/0	

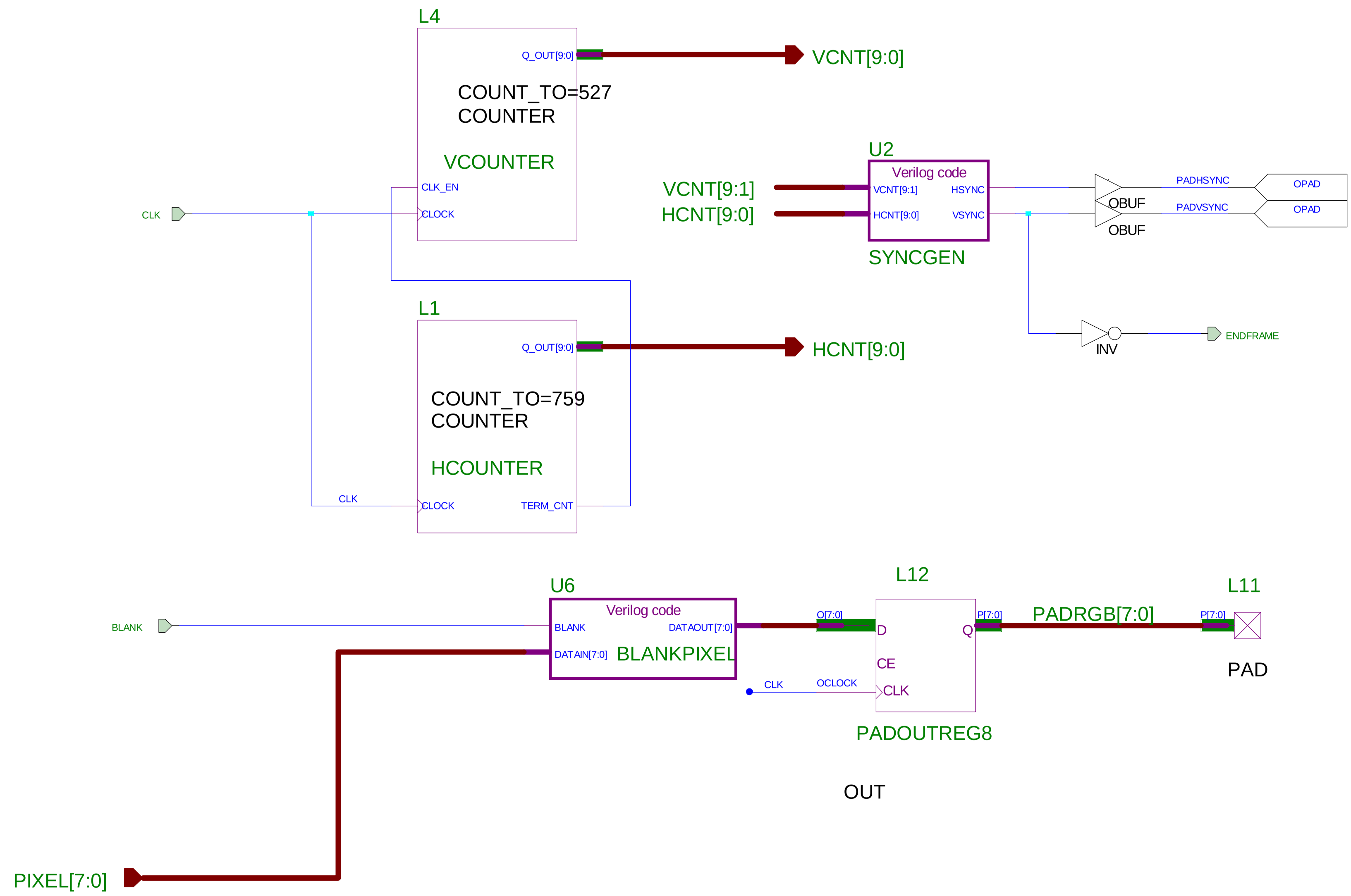


FIFO_128

Xilinx Corporation	Project: VGACAM
2100 Logic Drive	Macro: FIFO_128
San Jose, CA 95124	Date: 10/19/99
Date Last Modified: 1/17/0	



O17
FD
D
Q
C
CLK
OBUF
PADCS1
OPAD
INV
PADCS0
O15
PADADDR15
MD1
O16
PADADDR16
L7
MA[17:0]
MB[17:0]
O[17:0]
MUX
S
ADDRA[17:0]
ADDRB[17:0]
O[14:0]
L4
P[14:0]
PADADDR[14:0]
L1
CE
OCLOCK
OUTBUFREG15
PAD
U1
Verilog code
READA
BUSYB
WRITEB
PORTSELECT
WRITE_L
MEMCONTROL
READA
WRITEB
BUFGLS
This should be an AND with inputs and output inverted
OR2
PADWR
PADOE
L2
OE
O[7:0]
P[7:0]
PADDATA[7:0]
L3
WRITEDATAB[7:0]
TRIPAD8
OUTTRIREG8
READDATA[7:0]
I[7:0]
BIDI
MEM2PORT
Xilinx Corporation
2100 Logic Drive
San Jose, CA 95124
Date Last Modified: 1/17/0
Project: VGA
Macro: MEM2PORT
Date: 4/8/99



Xilinx Corporation	Project: VGACAM
2100 Logic Drive	Macro: VGACONTROL8
San Jose, CA 95124	Date: 10/19/99
Date Last Modified: 10/30/99	

```
// This module takes pixel values from the FIFO and writes the top left
// 256x256 pixels into the VGA memory space. We throw away the pixels
// outside this 256x256 area
// Memory asserts BUSY when it can't do the write - we repeat the write
// until BUSY is not asserted

module camera_test (Clk, Empty, Fst, Busy, addr_low, addr_high, Write, RE, frameEnd);

input Clk ;
input Empty ;
input Fst ;                     // frame start signal from the camera
input Busy ;                            // busy signal from memory
output [7:0] addr_low ;   // address to memory
output [7:0] addr_high;
output Write ;                          // write signal to memory
output RE ;                             // Read signal to FIFO - move to the next entry
output frameEnd ;               // Signals that we've reached the end of the frame

reg [8:0] hcnt, next_hcnt;              //horizontal pixel counter
reg [8:0] vcnt, next_vcnt;              //vertical line counter
reg Write, RE;
parameter LINESIZE = 351;       // Length of a line from the camera

assign addr_low = {hcnt[7:0]};
assign addr_high = {vcnt[7:0]};

wire inImage;
assign inImage =  hcnt < 256 && vcnt < 256;

assign frameEnd = (vcnt == 256);     // End of frame when we get to the 256th row

always @ (posedge Clk) begin
  if (Fst) begin                // We reset everything on Fst
     hcnt <= 0;
     vcnt <= 0;
  end else begin
     hcnt <= next_hcnt;
     vcnt <= next_vcnt;
  end
end

always @ (hcnt or vcnt or inImage or Fst or Busy or Empty) begin
    Write = 0;                  // Default values
    RE = 0;
    next_hcnt = hcnt;
    next_vcnt = vcnt;

    if (!Fst) begin // frame reset signal

// If the FIFO has something, and we are in the image, write to memory
                if(!Empty && inImage) Write = 1;

// If we had a pixel and memory was not busy, or we were not in the image, go on to next pixel
              if (!Empty && (!Busy || !(inImage))) begin
                        RE = 1;
                        next_hcnt = hcnt + 1;
                                if(hcnt >= LINESIZE) begin          // if we are at the end of the line
                                        next_hcnt = 0;                          // on to next row
                                next_vcnt = vcnt + 1;
                                end
                end
   end
end

endmodule
```

```
// This module assigns a memory bank to the camera and the monitor
// When the camera reaches the end of a frame, it is switched to
// the next frame and the next time the monitor gets to the end of
// a frame, it is switched to the next frame.  Since the monitor runs
// faster than the camera, it will display the same bank for multiple
// frames.

module bankselect (clk, cameraFrame, monitorFrame, cameraBank, monitorBank) ;

input cameraFrame ;             // Goes high when camera has finished a frame
input monitorFrame ;    // Goes high when monitor has finished a frame
input clk ;
output [1:0] cameraBank ; // Bank address used by the camera to store a frame
output [1:0] monitorBank ;// Bank address used by the monitor
reg [1:0] cameraBank ;
reg [1:0] monitorBank ;
reg [1:0] previousCameraBank ;          // Monitor uses the previous camera frame

reg  nextCameraBank ; // Control signal that loads cameraBank with next value
reg  nextMonitorBank ;  // Control signal that loads monitorBank with next value

// State register and state assignments
reg [1:0] state, nextState;
parameter cameraWait = 0, cameraDone = 1,
                monitorWait = 2, monitorDone = 3;

// Note that the state machine has no reset - it is self starting
// Note that we do not use delayed assignment when using previousCameraBank
// If we change the value in the first if statement, we want to use the new
// value in the second if.  (Currently this never happens because nextCameraBank
// and nextMonitorBank are never asserted at the same time.)
always @(posedge clk) begin
        state <= nextState;
        if (nextCameraBank) begin
                previousCameraBank = cameraBank;        // Save current camera bank
                cameraBank <= cameraBank + 1;           // Move to next bank
        end
        if (nextMonitorBank)
                monitorBank = previousCameraBank;       // Move to next bank
end

always @(state or cameraFrame or monitorFrame) begin
        // Default values for outputs
        nextMonitorBank = 0;
        nextCameraBank = 0;
        nextState = state;              // Stay in the same state by default

        case (state)
                // Wait for camera to get to the end of a frame
                // We don't come to this state until cameraFrame is turned off
                cameraWait: begin
                        if (cameraFrame) begin
                                nextState = cameraDone;
                                nextCameraBank = 1;             // Go to the next camera bank
                        end
                end
                // Camera is at the end of a frame
                cameraDone: begin
                        if (!cameraFrame) nextState = monitorWait;
                        else if (monitorFrame) begin
                                nextState = monitorDone;
                                nextMonitorBank = 1;
                        end
                end
                // Camera has started a new frame but we are still waiting for the monitor
                // to finish so we can move it along
                monitorWait: begin
                        if (monitorFrame) begin
                                nextState = cameraWait;         // we know cameraFrame = 0
                                nextMonitorBank = 1;
                        end
                end
```

```
module MemControl (ReadA, ReadB, WriteB, BusyB, PortSelect, WRITE_L) ;

input ReadA ;                        // Port A Read request
input ReadB ;                        // Port B Read request
input WriteB ;                       // Port B Write request
output BusyB ;                       // Port B request is not performed
output PortSelect ;          // Selects Port A or B to perform request
output WRITE_L ;             // Write control to memory

// This module does the arbitration between the two memory ports,
//    an A port, which only supports reads,
//    and a B port which supports both reads and writes.
// A request is made by asserting Read or Write along with the Address
//    (and Data if it is a Write request).
// The Busy signal is output on the same cycle as the request is made.
// If Busy is not asserted, then the request will be performed.
// Read data is returned on the *next* cycle after the request.
// A new request may be made on every cycle.
// Port A has priority over Port B: i.e. Port A can never be Busy.
// Reading Port B is the default operation even if not requested, so
// it doesn't have to be requested.  (This reduces the hardware at the
//    expense of increased power)

assign BusyB = (ReadA);                    // Port A overwrites Port B
assign PortSelect = !BusyB;                // Select PortB if it is not Busy
assign WRITE_L = !(WriteB && !BusyB);  // Write if Port B requests and is not Busy

endmodule
```

```
module syncgen (hcnt, vcnt, hsync, vsync) ;

input [9:0] hcnt ;
input [9:1] vcnt ;              // Bit 0 is a don't care, so we have to leave it out
output hsync ;
output vsync ;

// This module defines when the horizontal sync and vertical sync are asserted
// By asserting these syncs in the middle of the blank region, we are in effect
// centering the image displayed on the screen
// The horizontal count goes to 759
// The vertical count goes to 527

// There is a vile bug in the synthesis tools which throws away individual bits in
// busses attached to pins that are not used.  Xilinx says the bug is being fixed
// but for now we have to work around it

assign hsync = !((hcnt>=582)&&(hcnt<=674));      // These are magic numbers!
//assign vsync = !(vcnt>=490)&&(vcnt<=491);
assign vsync = !(vcnt==245);                      // A hack to avoid the synthesis bug

endmodule
```

```
module BlankPixel (DataIn, blank, DataOut) ;

input [7:0] DataIn ;
input blank ;
output [7:0] DataOut ;

// This passes the data from input to output, setting it to
// zero if blank is asserted

assign DataOut = (blank ? 0 : DataIn);

endmodule
```

```
            // Both the camera and monitor have finished their frames and have
            // moved to the next bank.
            // We need to wait for the camera to deassert cameraFrame
            monitorDone: begin
                if (!cameraFrame) nextState = cameraWait;
            end
        endcase
end
endmodule
```