


CSCE 611: Programmed I/O and Video Memory



Instructor: Jason D. Bakos



Lab 5

- Lab 4 extended until Friday
- Lab 5 was CPU synthesis, but we will merge this lab with the final project
- New grading structure:
 - Lab 1 (5%)
 - Lab 2 (5%)
 - Lab 3 (10%)
 - Lab 3a (5%)

 - Lab 4 (20%)
 - Lab 6 (30%)
 - Final exam (25%)



Input and Output

- CPUs need a mechanism by which to communicate with external devices, e.g. peripherals
- There are three primary mechanisms for this:
 - **Programmed I/O:** CPU and peripheral communicate through load and store instructions to special memory addresses
 - Very slow – used only for short “messages”
 - **Direct Memory Access:** an external device is used to allow peripherals to directly access system memory, without CPU involvement
 - Much faster – used for disk drives and network interfaces
 - **Interrupts:** special signals that peripherals can send to the CPU in order to notify it that an I/O, error, or timer event has occurred



Programmed I/O

- Loads and stores to specially-mapped address ranges can be used to:
 - Read a word from a **status register**
 - Used to poll the state of a peripheral
 - Write a word to a **control register**
 - Used to send an “instruction” to a peripheral
- In this course, your CPU must interface with:
 - VGA video memory
 - Switches
 - Buttons (“keys”)



Writing Control Registers

- Although real computers use DMA for sending data to video memory, we will treat each location in video memory as a control register
- TODO:
 - Expand data address bus from 10 bits to 16 bits
 - Memory addresses 0xC000 to 0xE000 are mapped to video memory
 - Instantiate video memory and VGA interface into “My Computer” design
 - Video memory is 8K x 15 bits
 - External hardware used to **mask off** DataWE signals and **write enable** video memory if store instruction falls into video memory range

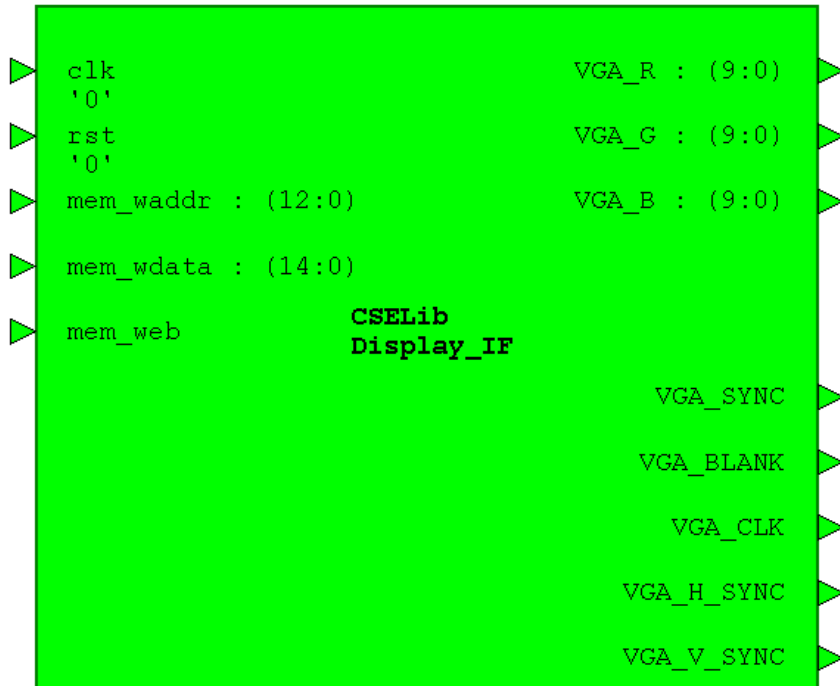


CSELib VGA Interface

- We have designed a VGA interface for you
- Our VGA interface has the following characteristics:
 - Output signal for 640x480 pixels frame
 - Each pixel is 30 bit color
 - 10 bits each for RED, GREEN, and BLUE
- Nominally, this would require 640x480x30 bits--just over one MB of data for a frame
- We don't have this much on-chip memory, so we will *downsample* the frame data:
 - The frame size will be 80 x 60
 - Our "pixels" will consist of 8 pixels wide, 8 pixels high
 - Each pixel will be 15 bits instead of 30



CSELib VGA Interface: Display_IF

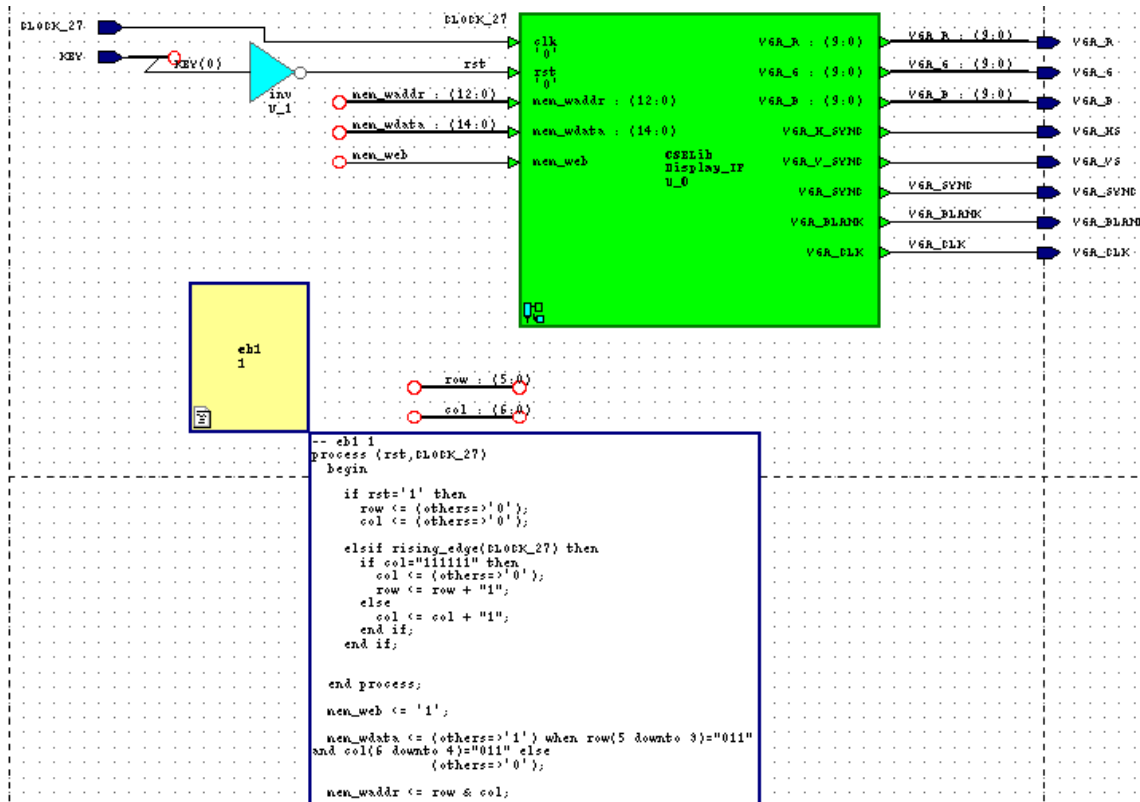


- The VGA interface outputs the current contents of the frame memory to the monitor
- To use:
 - Wire up clk, rst
 - To write a pixel:
 - Assert mem_web
 - mem_waddr(12 downto 7) is the ROW NUMBER
 - mem_waddr(6 downto 0) is the COLUMN NUMBER
 - mem_wdata(14 downto 10) is the RED intensity
 - mem_wdata(9 downto 5) is the GREEN intensity
 - mem_wdata(4 downto 0) is the BLUE intensity
 - The 8 outputs must be wired to output pins on the top level design, named according to their corresponding pin names



Example VGA Tester (w/o CPU)

- CSELib: Test_VGA
 - Places a white box on the screen

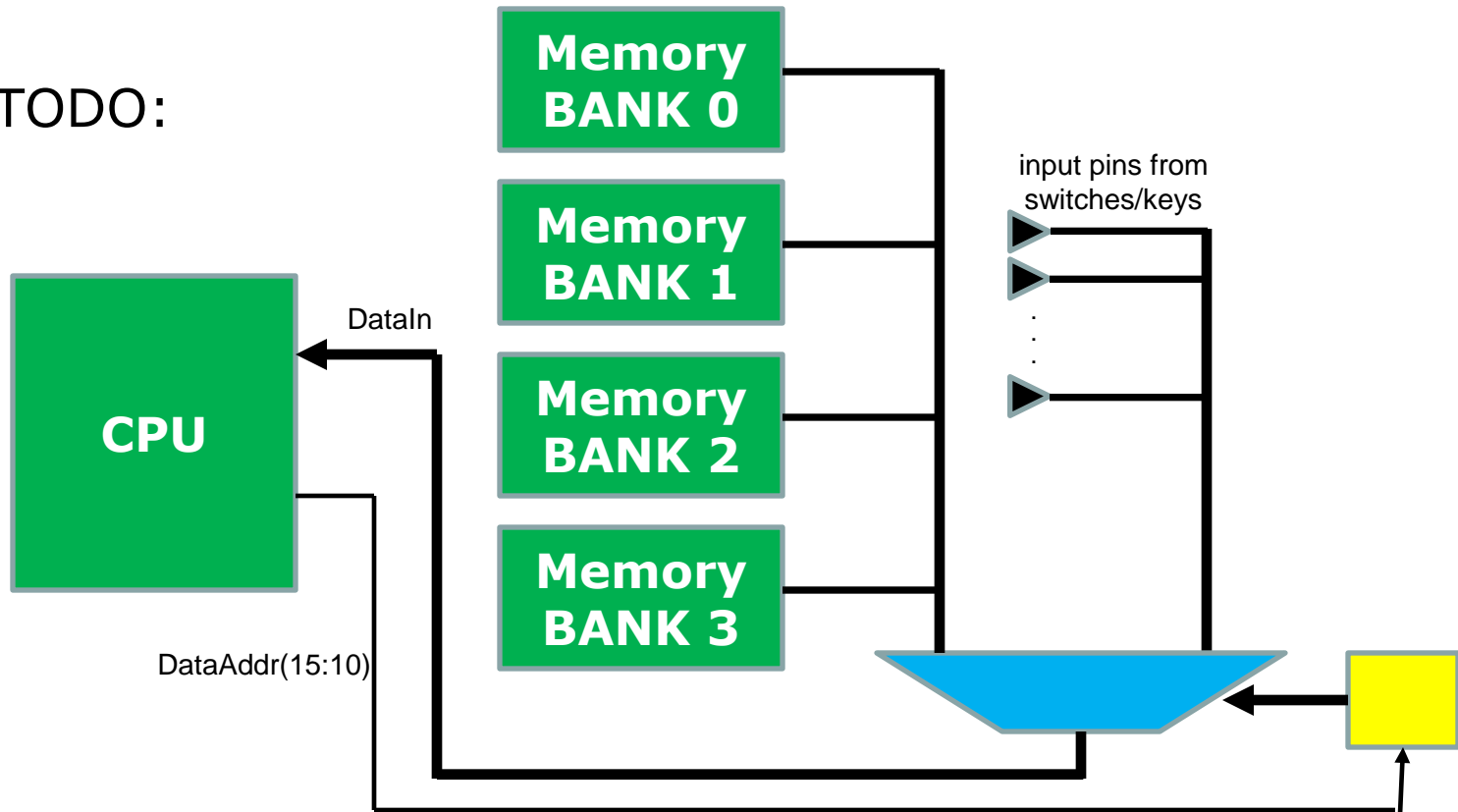


011000 <= row <= 011111
 24 <= row <= 31

0110000 <= col <= 0111111
 48 <= col <= 63

Reading Status Registers

- Associate the current state of all 18 switches and 4 keys with a single status register at memory address 0xF000
- TODO:



Your Goal

- Add these components to your “My Computer” design
- Write a new assembly program that implements a “bouncing ball”
- Use the keys and switches to apply a temporary force to the ball in four directions (up to two at a time)
- Controls:
 - SWITCH 0: controls the directionality of KEY 3 (left or right)
 - SWITCH 1: controls the directionality of KEY 2 (up or down)
 - Screen borders will stop cursor



Outline of Assembly Program

pos_x = INITIAL_X, pos_y = INITIAL_Y

- vel_x = 0, vel_y = 0
- loop:
 - reset forces (acceleration):
 - accel_x = 0, accel_y = GRAVITY
 - if pos_x = 0 or pos_x = 79 then accel_x = -vel_x * 2
if pos_y = 0 or pos_y = 59 then accel_y = accel_y - vel_y * 2
 - if button_left then accel_x--, if button_right then accel_x++
if button_up then accel_y--, if button_down then accel_y++
 - update velocity and position:
 - vel_x = vel_x + accel_x, vel_y = vel_y + accel_y
 - pos_x = pos_x + vel_x, pos_y = pos_y + vel_y
 - if pos_x < 0 then pos_x=0, if pos_x > 79 then pos_x=79
if pos_y < 0 then pos_y=0, if pos_y > 59 then pos_y=59
 - redraw frame, including borders
 - sleep for DELAY
 - use loop of TIME / [(#instructions+#stalls)*(1/27e6)] iterations



Outline of Assembly Program (GRAD STUDENTS)

pos_x = INITIAL_X, pos_y = INITIAL_Y

- vel_x = 0, vel_y = 0
- loop:
 - reset forces (acceleration):
 - accel_x = 0, accel_y = GRAVITY
 - if pos_x = 0 or pos_x = 79 then accel_x = -vel_x * 2 * LOSSNUM / LOSSDENOM
if pos_y = 0 or pos_y = 59 then accel_y = accel_y - vel_y * 2 * LOSSNUM / LOSSDENOM
 - if button_left then accel_x--, if button_right then accel_x++
if button_up then accel_y--, if button_down then accel_y++
 - update velocity and position:
 - vel_x = vel_x + accel_x, vel_y = vel_y + accel_y
 - pos_x = pos_x + vel_x, pos_y = pos_y + vel_y
 - if pos_x < 0 then pos_x=0, if pos_x > 79 then pos_x=79
if pos_y < 0 then pos_y=0, if pos_y > 59 then pos_y=59
 - redraw frame, including borders
 - sleep for DELAY
 - use loop of TIME / [(#instructions+#stalls)*(1/27e6)] iterations



Program Parameters

- On load, read from data memory:
 - INITIAL_X and INITIAL_Y
 - 32-bit unsigned integers
 - Address 0x0 and 0x4
 - GRAVITY
 - 32-bit unsigned integer
 - Address 0x8
 - DELAY
 - 32-bit unsigned integer
 - Address 0xC
 - LOSSNUM and LOSSDENOM (graduate students only)
 - 32 bit unsigned integers
 - Address 0x10 and 0x14



Project Due

- Project must be demoed to instructor and TA
- Each group must submit all design files and program to TA (jinz@email.sc.edu) by due date
- Due date: Dec. 9
 - Must be finished before final exam

Final Exam

- Individual, two hour practicum exam
- Friday, Dec. 9 at 5:30pm

