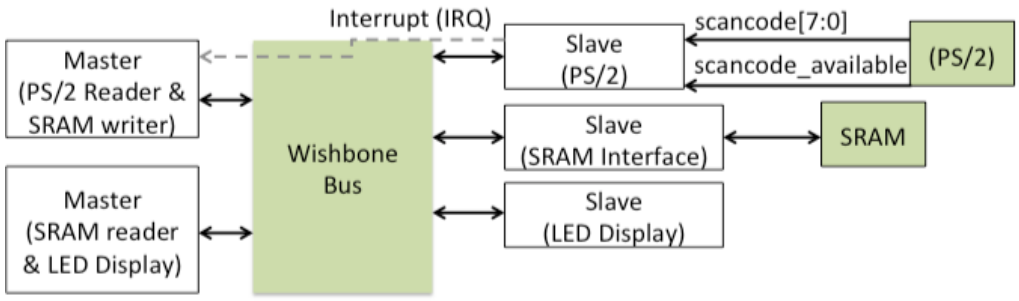
ECE 4401 (Fall 2014)
Lab 7 –Keyboard & SRAM Integration

The goal of this lab is to leverage lab5 and lab6 and create a system that can write text entered via PS/2 keyboard to the 7-segment LED display. You will create a module that will take input from the PS/2 keyboard and then write to a dedicated region in the SRAM memory. You will also create a separate module to read from the dedicated region of SRAM memory and display it on the 7-sement LED through the display module.

The SRAM memory region will mimic the 8-entry buffer from lab5. However, no shifting of memory contents is involved. The writer will start writing to SRAM at address x"000000" and increment by 4 for the next write. The buffer will be fixed to 8 entries, therefore, the writer needs to wrap around and start writing at address x"000000" once it has reached the end. The current location written to will be tracked by storing the address in the "head" register. The contents of only those 4 memory locations are retrieved that were written to most recently. For example, if one writes to buffer entry-0, entry-1, entry-2, entry-3, entry-4, and entry-5, only the last 4 memory locations (entry-2, entry-3, entry-4, and entry-5) will be read out and displayed.



The **wb_kb_read_sram_write** will wait for the interrupt from the keyboard interface module. Upon getting an interrupt, it will assert the cyc_o and stb_o signals to read the scancode data from the wb_ps2_kb module. When it gets the scancode, it will then pass that to the scancode2ascii module to convert it to an ASCII code. It will then send this ASCII value to the SRAM interface. This module will keep track of which memory location it is writing to by storing the memory location in a register, called "head". It will increment this address by 4, if the key pressed is not backspace. If the backspace key is pressed, it will clear out that memory location it last wrote to by writing x"20" to it, and decrement the "head" pointer by 4. This module will write in a range of memory addresses and then wrap around when it reaches the end. The starting address of this memory range is x"000000" and the last address is x"00001C". Rest of the addresses are increments of 4 (x"000004, x"000008" …). This mimics an 8 entry circular buffer.

The **wb_sram_read_segled_write** will continuously try to acquire the wishbone bus. Once it has control of the bus, it will read out the last 4 memory locations (4 separate accesses) that were written to. This module starts the first access from the address pointed to by the head pointer, and then decrements that address by 4 for the subsequent accesses. This module keeps on storing the returned value in a local register. Once it has all the data, it sends that data (32 bits) to the display module. The display module is hard coded to display the most significant 8 bits on the left-most 7-segment LED display and the least significant 8 bits on the right-most 7-segment LED display.

The **wb_kb_read_sram_write** module should initially take control of the bus by asserting cyc_o (**wb_sram_read_segled_write** should delay asserting its cyc_o by 1 cycle). It gives up the bus by de-asserting cyc_o for 1 cycle, after it is done writing to the appropriate location in the SRAM. At this point the **wb_sram_read_segled_write** will take control of the bus and perform 4 SRAM read operations, and store the returned ASCII value in a register. It will then send this data to the display module and give up the bus.

Notes:

1- Most of the code is either provided or being reused from lab5 and lab6. You can use the exact functionality of your keyboard state machine in **wb_kb_read_sram_write**, and instead of sending it to the display module you now send it to the SRAM (lab6 SRAM **writer** functionality). Similarly, you can reuse the SRAM **reader** functionality form lab6 in **wb_sram_read_segled_write**.

2- You need to make sure that the **wb_kb_read_sram_write** takes control of the bus. It can be achieved by asserting cyc_o on reset while de-asserting cyc_o on reset in **wb_sram_read_segled_write**.

3- You need to make sure that the **wb_kb_read_sram_write** module gives up the bus for 1 cycle to give a chance to the other master to get control of the bus. This can be achieved by de-asserting cyc_o in a state after the write to SRAM has finished.

4- The **wb_kb_read_sram_write** module needs to clear out the 8 entry buffer by writing x"02" to each address at the very beginning. This needs to be done so that we don't have any garbage values being displayed. This can be achieved by going to a state after reset and performing the 8 writes and then moving on to the main states, never coming back to this state.