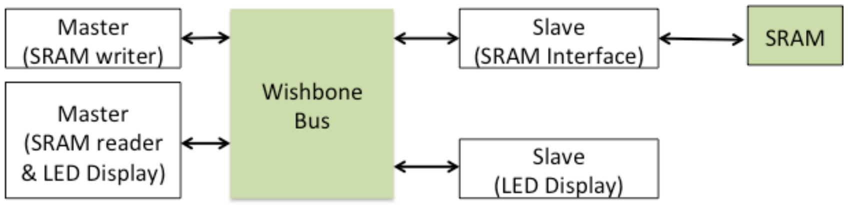
ECE 4401 (Fall 2014)
Lab 6 – SRAM Interface

For this week's lab, you will need to create three new Wishbone modules – a memory writer, a memory reader, and extend the provided SRAM interface module.



The SRAM interface will translate requests from the Wishbone bus and present them to SRAM memory. The SRAM is organized as 8 million words of 16-bits each. Thus, the total amount of available SRAM memory is 16 mega bytes. The SRAM is addressed by a 23-bit address bus (MemAdr(23 downto 1)) and a 16-bit data bus (MemDB(15 downto 0)). RamCS enables the SRAM, and the RamUB and the RamLB bits are used to select the upper or lower bytes of the 16-bit word. In addition, MemWR is used to indicate a write and MemOE is used to indicate a read. Note that all control signals are active low. The SRAM can be operated in a variety of modes that are controlled by the RamAdv, RamClk, and RamCRE signals. These should all be set to "0". Since the internal Wishbone data bus is 32-bits wide, the SRAM interface will need to read and write the SRAM in two transactions. Pages 13-14 of the FPGA board manual (http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf) have more details on the organization of the SRAM.

The memory chips that are used are the Micron MT45W8MW16 *70ns* SRAMs. In summary, when doing a read, the data is available *70ns* after the address lines have settled. For a write, the address must be stable for *70ns* before the write is complete and the data must be stable for *20ns* before you can change the data. Since the FPGA is running at 50 MHz (*20ns* period), the SRAM interface must be extended to insert appropriate wait states when doing a read or write.

The **SRAM interface** will communicate with the bus and the physical SRAM. You will extend the provided module by designing a state machine that does the communication with the bus and the SRAM. The state machine will also need to take care of the wait cycles, when communicating with the SRAM. As the SRAM data bus is 16-bits wide, the data received from the wishbone bus will be written to SRAM in 2 accesses. First the lower 16 bits and then the upper 16 bits will be written. Similarly, 2 accesses will be made to retrieve the 32 bits of data on an SRAM read. The state machine needs to take care of the wait cycles (at least 4 cycles) for each individual access.

The **SRAM writer** will continuously write to memory by incrementing the address. The data it should write to memory should be the address. Thus, write "00000000" to address "000000", "00000004" to address "000004" and so on. The memory writer will be a module on the Wishbone bus. Note, that since the default Wishbone bus arbiter does not guarantee fairness, you will need to write code in the memory writer module to force it to release the bus so that the writer does not take over the bus and not allow the reader module to get access to the bus.

Finally, the **SRAM reader** will continuously read from memory and display the data on the 7-segment display. Since we need to be able to read the 7-segment display with a human eye, you will need to slow down the reading module so that it can display a new piece of read data every 2 seconds. For one second, the display will show the high-16 bits of the 32-bit data, and the next second it will show the lower 16 bits. Thus, the memory module will be reading from the SRAM module and then writing to the 7-segment module.

The library modules you need for this lab are,

- clock_divider
- wb_intercon
- wb_arbiter
- wb_segled
- word2leds (which uses hex2led and char_led_control)