**The CPU and Datapath**

The CPU of the T.E.S.S. is based on the CR16 ISA with some modifications as to which instructions were really necessary. Instructions such as multiplication were removed to ensure that the T.E.S.S chip would fit in the required space. A clock speed of 12.5MHz will be used, although the chip could utilize a faster clock, the 12.5MHz clock was chosen to simplify VGA implementation. For more detail as to the reasons behind the chosen clock speed see the section of the report discussing the VGA implementation.

The CPU was designed to be integrated with off chip memory due to the area constraints for fabricating the chip. Specifically, both the ROM and SRAM for the system are off chip. This facilitates our goal of developing a gaming system similar to a SNES or NES, as the instructions and glyph library will contained on the ROM. Switching games will be similar to switching cartridges in a SNES, but instead consists of swapping ROM chips with preloaded games. Due to the number of pins available on a 4 tcu chip, 78 total signal pins, our SRAM and ROM share a 16 bit data bus and a 16 bit address bus. Currently the design consists of 16 bit data in bus, a 16 bit data out bus, and a 16 bit address bus. This will be changed before fabrication as the SRAM chip selected shares pins for data in and data out making a bidirectional data bus necessary. This will be more easily managed on chip than off and will need to be corrected before final fabrication.

The ROM will contain both the glyph library and the instructions. The ROM must be accessed by both the CPU for instruction fetching and the VGA controller for access to the glyph library. The SRAM is also partitioned to contain the frame buffer for the VGA. Additionally using memory mapped I/O (MMIO) results in a section of the SRAM that is not accessible due to the addresses being reserved for our controllers, audio, and additional I/O. Again the SRAM is accessed by both the CPU and the VGA controller. Two 8x8 glyphs are stored per 16 bit word in the ROM. This gives a total of 256 glyphs available for our applications. Using 2-bit color and a 16 bit word SRAM enables the entire row of pixels to be stored per word. Every 8 clock cycles the glyph library must be accessed to determine which pixels are needed for the glyph because there are 8 pixels per glyph row. It takes 2 clock cycles to retrieve the glyph information and display it. This leaves 6 clock cycles available for the CPU to access memory. Every instruction in the design requires 2 clock cycles, for a fetch and decode/execute state. This enables the CPU to run 3 instructions before it must wait for the VGA to display again.

The CPU operates by reading an instruction from the ROM. Once an instruction is fetched from ROM the logic controller decodes the instruction, and sets the appropriate control signals based on the instruction. The logic controller sends the corresponding ALU opcode, and decodes which register file addresses are required if any. Accessing the off chip ROM and SRAM requires a memory controller to ensure only SRAM or ROM is enabled at a time. The memory controller sets the chip enable for the ROM and SRAM, and also sets the write enable for the SRAM when performing a store instruction. ROM is only accessed to get the next instruction on the update of the program counter, and for the VGA controller to fetch a glyph from the glyph library every 8 cycles. SRAM is free whenever the CPU can execute instructions leaving it free for load and stores.

The CPU was designed in Xilinx with the intention of implementing the project on FPGA before fabrication. Xilinx was great for simulating with this in mind, but it takes issue with using tri-state buffers for bidirectional buses. Therefore, the CPU was designed with MUX’s instead. This leads to a problem in changing our data in and out lines for the SRAM since these buses need to be bidirectional the design will be tested in ModelSim to correct for the shared data in and out pin. The SRAM and ROM that reside off chip were modelled in Xilinx and should operate well within the timing constraints. Finally the chip was synthesized.

After synthesizing the core with a register file implemented in Verilog it was decided that utilizing the provided SRAM register file would be more effective to keep the chip with in the desired area. The CPU was updated at this point to be compatible with the provided register file. Utilizing this 16x16 register file was not much different than the one we simulated with in Verilog, but additional testing will be needed before fabrication to ensure that everything is operating as intended.

Our first synthesis attempts went well in EDI. We were able to synthesize the VGA controller, audio, and CPU. At this point the individual blocks; audio, VGA controller, CPU, and register file; were placed in the Layout Suite XL for ccar routing. After using ccar the system passed DRC but continued to fail LVS despite our efforts. Alas it was determined that the ccar was not able to route the large numbers of pins from each of the modules. To remedy this shortcoming of the ccar tool the entire system, except the register file, was implemented in Verilog. EDI was then utilized to generate the VGA controller, audio, and CPU in one circuit. The register file, as well as the pad frame, was then routed to the rest of the system using the ccar tool. At long last the T.E.S.S. was constructed in all its glory and passed DRC and LVS. It was decided at this point not fill the chip with the required material densities for CMP during fabrication due to the minor changes and additional testing needed. As a team we plan to finish testing and modifications over the break, and seem to be on track for fabricating the chip called T.E.S.S.