CAD9 Peripherals and Top-level Integration

Assignment

Design peripherals or additional functional blocks for your project. Complete final chip assembly, functional and physical verification. In short - finish your project!

Description - Peripherals

A microcontroller design attempts to realize a system on a chip, eliminating, to the extent it is practical, the need for any other integrated circuits. In cases requiring large memories, it may not be practical to put all of the memory on chip, and in our 427 projects, we are not including analog blocks on chip. But in most cases, all of the I/O functionality needed for the target application should be on the microcontroller. As a simple example, if the microcontroller is to work with a keypad and an LCD display, the keypad should not require any active circuits, and the display should be driven directly from the microcontroller.

Most of your projects will need to incorporate one of the following peripheral devices: parallel I/O (for A/D or D/A interface, solenoid control, contact-closure input or output, etc.), serial I/O (asynchronous or synchronous), programmable timer, keypad controller, display driver, mouse input, printer output, floppy or hard disk interface, or special arithmetic functional unit. You must generate the control signals with proper timing for any of these interfaces.

Multipliers, MACs or other units that are more tightly tied to the instruction set architecture are usually part of the core architecture and are not peripherals per se. They do not have any sort of interrupt interface wiith the core's controller/datapath but are accessed directly via instructions in the ISA. You are free to use full-custom or synthesis/APR design techniques (or a combination of the two) for this assignment.

Description - Top-level Integration

Your top-level schematic should contain all of the following blocks: controller, datapath, peripherals and i/o pads. Note that you should place internal memories in a separate, higher level of hierarchy as explained in class. Once you have thoroughly verified this schematic in Modelsim, then you must complete the top-level layout and DRC/LVS.

Procedure - Peripherals

Since your applications are very different, each group's work will be unique for this portion of the assignment. One feature of your work that may be similar is that you might use memory-mapped I/O, as defined by the instruction-set architecture. This means that data is read from peripheral devices using a LOAD instruction, and written to peripheral devices with a STORE instruction. The addresses for these peripheral devices are in a section of the data memory address space which is not used for memory. You must decide which space you are reserving for I/O devices. For example, in a system which needs only a little data memory, you could take the full top half of the address space by looking at only the high-order bit of the address. When it is a 1, you know that the LOAD or STORE instruction is a read or write command to a peripheral device, rather than to memory. Assuming there are multiple peripheral devices, you have to further decode the LOAD or STORE address to determine which device is to be selected. When I/O is being done, it is mandatory that you disable the control lines to data memory which would otherwise erroneously write to memory or turn on tri-state buffers from the data memory, causing contention with the tri-state buffers driving the result onto the internal write-bus from the peripheral device. This can be accomplished by controlling memory chip- and peripheral-select lines which are glitch-free and have at most one active line at a time.

A microcontroller designer has considerable latitude in deciding how dumb (simple) or intelligent the peripheral devices will be. Peripheral devices quite commonly have their own configuration registers and status registers, and are controlled by state machines. One can, however, put more of

the burden on the microprocessor if it has time to take care of low-level peripheral activity.
The interface between the processor and peripherals can be managed either by having the processor poll the status of the peripheral, or by providing an interrupt structure which forces the program counter to the beginning of an interrupt service routine when the interrupt has new input data or is ready to take more data for output. An interrupt system must have the ability to disable interrupts during critical program sections, to save and restore the status of the processor upon entering and returning from an interrupt service routine, and in most cases, the ability to prioritize interrupts. Direct memory access is another very common peripheral function which unloads the processor at the expense of some special-purpose hardware. DMA allows peripheral devices to stop the processor and take control of the buses. The DMA controller can generate all of the control signals to move data directly from the peripheral to or from data memory.

Procedure - Top-level Integration

Build the chip-level schematic, instantiating pad symbols from $TSMC25/parts/pads/schematic. Be sure to include pads for vdd, gnd and corner cells. Include at least one vdd/gnd pair for each side of the chip. You will need four corner pads. These have no pins and therefore no real netlist purpose, but you can include for completeness. You will not be able to perform LVS on a layout containing the on-chip memories since we do not have schematic representations of these blocks. You should therefore bring all of the signals in the memory interface out to ports and place the memory in a higher level in the schematic hierarachy (layout too). You will later run DRC on the full chip (including memories) but LVS on the level that DOES NOT include the memories. Then verify your design in Modelsim completely, testing all peripheral interfaces, the test program, your own test programs, test modes (e.g. scan chains) and any extra instructions you've implemented.
When you are satisfied with the functional verification, you are ready to start on the layout. The top-level layout design must be completed manually in IC-Station. The top-level APR flow is not working at this time. See the comments below for information on how to go about doing this layout.
If you have internal ROM, you do not have to be concerned about generating a programmed ROM unless you are fabricating.

Comments

- You don't have to show your entire application working in Modelsim. But you should test out each interface (core-peripheral, peripheral-external) such that you are confident all interfaces would work together in the final system.
- The top-level physical design will be tedious. First, place all blocks except the i/o pads. Estimate the number of routes you will have to run in each channel, along with estimates for power, ground and clock routes that may be wider than minimum width. Then add some extra space to leave room for signals you neglected or other unforseen problems. I am not concerned about wasted space here and there in these channels. What you want to avoid doing is leaving the channels too small, then going back and ripping out a lot routing and starting over. You might consider making some via cells to facilitate the routing effort. These would be composed of each layer (metals and vias) to create a DRC clean via connection. You should also learn to use path routing entry if you haven't done so yet. Paths are fixed-width shapes that can be added more quickly than shapes. For paths, you only need to specify the turn-points in the path route, not every vertex. After adding the path, you can change the width quickly with "cha path" <return> then type the desired width and <return> again. If necessary, you can flatten a path into a shape by selecting it and Edit->Flatten. The last step is to put the i/o pads on.
- Ideally you can finish the Modelsim verification prior to starting the top-level layout. On the other hand, it might be a good idea to get someone looking at the layout early on since it is a large task. If layout begins prior to final Modelsim, be sure to take into account possible late changes due to bugs popping up in functional verification.
- Leave a good amount of time to complete top-level LVS. LVS at this level can be very painful to debug.

Requirements

- If your peripherals contain any full-custom schematics, include proper delay information (delay boxes) for those portions of the design.
- Full-chip schematics. If you have internal memories, put them on a separate schematic containing a symbol for the rest of your chip. Run modelsim on this top-level schematic and run LVS on the sub-schematic that doesn't have the memories (this is compared with the layout with the memories removed. Other than this, you should be using the same schematic database for LVS as you're using for modelsim. This is a key point in verifying your design.
- Verilog or schematics for all peripherals or functional units.
- Modelsim traces for $TSMC25/misc/assembler/testfile.asm.
- Modelsim traces showing proper functionality for the peripherals. You should simulate the peripheral or functional unit at the top-level, not stand-alone. The simulation should show any important aspects of the core/peripheral interface (interrupts or software polling, data transfers). Each interface should be tested: core-peripheral, peripheral-external, peripheral-peripheral, etc.
- Modelsim traces for any test modes you've implemented.
- Modelsim traces for any extra instructions or features you've added.
- Testbenches or models of external devices you've used for each type of verification.
- Layout of the complete chip (including on-chip memories).
- DRC and LVS reports for the complete chip.
- README containing a good description of your modelsim verifications. Describe what is happening in each simulation (testfile not important here). Describe all top-level signals (very briefly -just enough that I can figure out what's going on when reviewing your design). This includes pads and any internal nets at the top levels of the schematics. This means you should give net names to every signal at the top-level (this will help in LVS too). As usual, includes paths to all relevant files. Also, provide enough information for me to be able to run Modelsim runs of my own. This means you must describe any instruction sequences that are for some reason unsupported or any tweaks you must make for some reason to the output of Export Verilog.
- Final Demonstration. I will sit down with each group to look over your final Modelsim runs, chip layout and full-custom blocks. You should be ready to discuss everything that is required for this CAD assignment and any other project-related issues. You don't need to do any special preparation for this (i.e. don't review old CAD assignments, etc.) but plan to spend an hour. Try to have relevant simulations, schematics and layouts up and ready to look at. If there are problems or there are things you were unable to finish, you should tell me about those. It is better for you to tell me about those than for me to find them on my own when I review your design in more depth outside of the demo. This is also a good time for you to highlight any aspects of your design that you think went particularly well or those that didn't go so well and how you might have done things differently.