**This outline is intended as extra helpful notes and is not comprehensive.**

**Combinational Logic Basics:**

* 1. Signal restoration and its importance in digital circuits.
		1. DDCA 1.6
	2. Function of primitive logic gates.
	3. Derivation of truth tables from simple combinational logic circuits.
		1. Create table
			1. Label your truth table as binary inputs 0 to (N-1).
			2. If necessary (for larger circuit), label intermediate signals and expand your truth table to include them.
		2. number of Boolean functions with N inputs, M outputs
		3. 7LUT from 6LUTs
	4. Operation and implementation of multiplexors.
		1. implementation with logic gates or transmission gates
1. **State Elements and Sequential Circuits:**
	1. Basic flip-flop operation. Flip-flop Input/output/clock timing and constraints.
		1. clk-to-q delay; setup time constraint; hold time constraint
	2. Flip-flop reset types and functions.
		1. asynchronous reset/set: timing and how to implement
		2. synchronous reset/set: timing and how to implement
	3. "Register transfer" notion of sequential circuit timing.
	4. Waveform diagrams
		1. practice with a few different circuits and inputs
		2. including the clk-to-q delay in the diagram can sometimes make things clearer
	5. Pipelining and signal feedback.
		1. accumulator circuits, counters
		2. counter/decoder pattern
		3. pipelining can decrease combinational logic delay (and allow you to increase clock frequency) but increases latency of an operation. However, the operations/cycle of circuit can remain the same as long as operations overlap in the pipeline (e.g. decode instruction while executing previous instruction
	6. Level-sensitive latch operation. Implementation of flip-flops using level sensitive latches
		1. low-level sensitive latch output into input of high-level sensitive latch
	7. Flip-flop operation of Virtex 5 FPGA.
		1. optional FF on output of every LUT6
	8. LUT implementations, shift registers
2. **Verilog**
	1. cheatsheet ideas: examples (for syntax), FSM parts, rules
	2. structural
		1. wires and gates
		2. NO dataflow operators (+, ^, |, & etc)
	3. behavioral
		1. always @
			1. posedge Clock – inferring state
			2. \* - inferring combinational logic
		2. assign
			1. inferring combinational logic
	4. assignments
		1. use non-blocking (<=) in always @ posedge
		2. use blocking (=) in always @ \*
	5. wire/reg
		1. don’t forget input/output declarations
		2. reg if left side of an assignment in always block
		3. wire if not
	6. generate and parameters
	7. case versus if-else
		1. if-else will create priority logic if the if-conditions are not mutually exclusive
		2. make sure every path through the always@\* block assigns every signal that has any assignments in that block (to avoid unexpected latches)
			1. understand why latches are inferred when this happens
	8. simulation
		1. Discrete event simulation; “Simulation” lecture
	9. synthesis
		1. inferring hardware
		2. steps from “Synthesis” lecture and LabLecture2
	10. FSM in verilog
		1. suggested parts:
			1. state encodings (localparam)
			2. state-transition (always@(posedge Clock))
			3. next-state logic (usually always@\* with case)
			4. output logic (always@\* or assign)
	11. testing
		1. comprehensive, random with hardcoded corner cases, random-targeted