Topic (alphabetical)	First digital logic + lab	Digital logic (advanced)	First programming	Microprocessors	Embedded Systems	Networking	Organization & Architecture	Algorithms	General
abstract thinking and modeling	0,1 can mean yes/no, enable/disable, true/false, numbers ….; 1+1=1?		algorithm-flowchart-code	binary string as instruction has a different meaning from same binary string as data	programming models; relation to underlying hardware	levels of protocol stack; encapsulation	ISA abstraction	abstract models (including programming models), proofs, analysis	Abstraction; "information = bits+context"
addressing modes	see cycles/instr		pointer use	main exploration of topic	content addressability		address vs data		
alternate representations/conventions (for convenience/advantage)	Truth table = K-Map; state table = state diagram		different data structures for same data (ex: incidence matrix, edge list for graphs), pre, post and infix	different representations of same data (binary-hex, unsigned, BCD)			Ex: left/right endian	different data structures. Modeling tools	
Amdahl's Law	carry propagation as a bottleneck in ripple carry adder	critical paths						parallel algorithms	bottlenecks
Arithmetic	arithmetic circuits (adders and perhaps multiplier, divider). Number representations			possible efficiencies (stemming from proper selection of operation/instruction), overflow			ALUs, possible efficiencies, stemming from proper selection of operation/instruction; Strassen's matrix multiplication		
assembly vis-a-vis hardware	Relationship between hardware, assembly/machine code and high level code			programming models; relation to underlying hardware			programming models; exploiting efficiencies		
Complexity and Analysis	Digital logic can provide a first hand look at complexity. It is intimately tied to circuit complexity and is replete with intractable problems. For example, state assignment could easily illustrate the futility of an exhaustive approach		complexity of algorithms, program efficiency		CAD tool use can provide a window to appreciating the importance of complexity in practice	Can provide a setting for probilistic and amortized analyses		Complexity theory	
Control structures (conditional, loop)			introduction at HLL	implementation in assembly			implementation in assembly		
delay	circuit delay	area/time optimization; retiming, clock skew	program speed	instruction delay	Ex: FPGA clocking rate	latency	instruction delay,pipeline delay, access time	algorithms as a general recipe, implementable in hardware or software	see performance measures
distinction between control and data	Example in MUXs, ALU etc. (data width is independent of control width). Complex circuits with control FSM enabling the operation of "data" part		control instructions		control and data hazards		indepencence in width of control and data paths		
floating point add/mult, rounding	introduction to idea that n bits can only represent 2^n distinct values	implementing floating point hardware	introduction of float	Details of floating points/standards	precision, errors	standards	instructions	precision, errors	Standards
Graphs	state diagram as a graph, can relate binary numbers to graphs	place and route, connectivity, planarity	data structures (trees, linked lists)		DAGs, task graphs	routing and congestion control		Graph algorithms	see also modeling and abstraction
hazards	logic hazards, reconvergent paths, synchronization using registers, async gates in synchronous ckts,				synchronization, handshaking		data, structural and control hazards (branch prediction),out-of-order execution, race conditions, synchronization, deadlock, livelock		
idea of "state"	From flip-flop outputs to "what needs to be remembered"		need for memory in computation (example swapping values)	CPUs as FSM	Checkpointing and state restoration; FPGA pattern matching constructs a state machine for the pattern (such as in the KMP algorithm)		Checkpointing and state restoration (particularly in distributed environment); oblivious (limited-state) computation		
instrction (cycles per instr, format,encoding)	can be introduced with circuit delay in an advanced example (that implements an instruction or an address computation)		instruction choice (example shift or multiply)	cycles/instr, instruction choice (example multiply by a constant), format	constant coefficient multipliers (via look-up tables)		instruction choice, ISA, RISC/CISC	see arithmetic	
interrupts/exceptions	enable, asynchronous inputs; event triggered Verilog execution			interrupts and exceptions, priorities				event-driven processes	
low-level paralleism	bitwise boolean operations as opposed to scans and global operations			instruction implementations	Low-level optimizations (for example in FPGAs)		ILP, supersacalar		

Topic (alphabetical)	First digital logic + lab	Digital logic (advanced)	First programming	Microprocessors	Embedded Systems	Networking	Organization & Architecture	Algorithms	General
memory	memory as an abstract idea to explain state	memory as a circuit	memory as a variable	memory as registers and RAM, different types of memory			memory hierarchy and management; memory architecture (shared, distributed); performance	memory as an algorithmic abstraction (online, oblivious)	
Models/Structures of interaction/communication	The idea of interconnects and topologies can be introduced in the first digital logic	bus (and other stucture) implementation	parameter passing, shared variables	data and control buses, interconnects and interfaces		client-server, peer-to-peer, routing, flow control, protocols and services between protocol layers	shared/distributed memory, message passing, routing,interconnection networks, topologies		also see abstract thinking
Modular design	Verilog modules, circuit decomposition		procedures, scope of variables	bit-slice	Verilog modules	layers in protocol stack, encapsulation			See also abstraction
	course sequnce itself may be viewed as an exercise in modular exposition of concepts, for example, electronics --> digital logiv --> microprocessors --> embedded systems --> computer organization --> Computer architecture								
parallel vs. sequential	First look at parallel. Circuits are inherently parallel. Many possibilities, serial to parallel conversion, carry look-ahead, barrel shifter, Verilog blocking and unblocking assignments etc.		First look at sequential		multiple pieces running sequntial code independently	Sequential/Parallel at different layers (Transport layer may deliver packets sequentially (in order), network carries packets in parallel paths/links. (see also abstraction)	Multicore environment ...	parallel, sequential, distributed algorithms, concurrency	
performance measures	delay, number of gates/flip-flops	clock-speed, area, power	time (asymptotic)	processor cycles, memory	speed, power, memory, cost/form-factor	latency, throughput, quality or service	speed, power, fault-tolerance	time, space, communication cost, approximation ratio	
pipelining concept	series of flip-flops (example shift register) as an analog of progess through an ideal pipeline	pipeline implementation			Ex: image processing pipeline	pipelining packets across (frame-level) links; sliding window protocol	instruction pipeline	sofware/algorithmic pipelining	
procedures, parameter passing, scope of variables, run-time stack (see recursion also)			procedures, stacks	details of parameter passing, runtime stack		Remote Procedure Call (RPC)			see also Modular design
recursion (see procedures also)	Decompsition of MUXes, decoders etc. Carry lookahead as a recurrence	recursive harware blocks (example bitonic sorter), prefix circuits (ex: Kogge-Stone)	recursion, recursive structures (such as trees)				Recursive structures	Divide and Conquer	Recurrences
shared resources	MUX to share output among several inputs			Shared buses, hardware modules, hardware reuse (FPGA)		MAC layer, multiaccess	shared resources (memory, channels, processing elements), deadlock, semaphore, critical region		Multiplexing in other dimensions (besides time), for example frequency/wavelength
speedup	obliquely through "parallel" examples. How much is delay reduced by increasing H/W cost	area/time optimization						parallel algorithms	see tradeoffs, performance measures
Synchronization	latch, flip-flop, circuit timing				handshaking between modules	flow control, handshaking (protocol level), traffic shaping (example leaky bucket)	interfacing between modules (example CPU-RAM)	distributed systems, I/O streams and buffering, concurrency	
throughput	"throughput," for example in series parallel conversion					network throughput			see performance measures
Tools	physical device vs CAD tools		compiler, debugger	simulating, debugging, verification, testing (in different contexts)					
Trade-off	delay-area		speed-space (array-linked list)		speed-area(cost)-power		performance tradeoffs in memory hierarchy, interconnect density, scalability	space-time-communication complexity	
translating informal specs. (see also abstract thinking and modeling)	verbal/informal description to state diagram/architecure		verbal description to algorithm/flow-chart					Problem to algorithm	
verification and testing	HDL testbench, verification flow		proof of simple programs, loop invariants, induction (see also recuirsion)		HDL testbench, verification flow	protocol verification (safety, liveness)	Architecture/Software verification/testing; algorithm correctness		