DSP Design Methodologies and Tools

Proudly written by Y. Wu with neither confidence nor guarantee!

§1 Introduction [1-8][1-59]

1. Characterise embedded systems and give examples of application areas.
    - Sophisticated functionality
    - Real-time operations: Hard/soft realtime
    - Low manufacturing cost: limited memory, uP power
    - Low power: dynamic voltage/frequency scaling, software design

2. What are the advantages of using microprocessors for embedded system implementation ?
    - Flexible: can use same logic to perform many different functions
    - Simplify the design of families of products
    - Much more logic than **Custom Logic** low power winner!
    - CPU: heavily pipelined, large design teams, latest VLSI tech, lower efficiency

3. What are the major challenges in embedded system design?
    - Hardware: CPU Memory
    - Deadlines: faster hardware or cleverer Software
    - Power: Turn off unnecessary logic, reduce memory accessed
    - Work?: correct specification, implementation, real-time, data test?
    - Work on system: observability, controllability, development platform

4. What is meant by a “design methodology”?
    - A procedure for designing a system.
    - Helpful: Compilers, Software engineering tools, EDA tools

5. Compare top-down vs. bottom-up design methodologies.
    - Top-Down: Start from most abstract description to most detailed
    - Bottom-Up: Work from small components to big system.

6. How can system requirements be captured?
    - Functional / Non-functional Requirements.

- name, purpose, inputs, outputs, functions, performance,
- manufacturing cost, power, physical size/weight

7. What is contained in a specification?
   - A more precise description of the system: should not imply a particular architecture, provides input to the architecture design process
   - May include functional & non-functional elements
   - May be executable or may be in math form for proofs

8. Define “architecture design” and “system integration”.
   - **Architecture Design**: Components(H/S) to specification, incl. functional and non-functional specs.
   - **System Intergration**: Put together the components (many bugs), have a plan for integrating components to uncover bugs quickly, test as much functionality as early ap.

§2 System Design [9-21][60-106]

1. What is a **design flow** and why is it important?
   - Sequence of steps in a design methodology.
   - Maybe partially or fully automated (tools to transform & verify design)

2. Describe the **waterfall** model and its pros and cons.
   - Requirements: determine basic characteristics
   - Architechture: decompose into basic modules
   - Coding: implement and integrate
   - Testing: exercise and uncover bugs
   - Mantenance: deploy, fix bugs, upgrade
   - **Local** feedback only: may need iterations between coding & requirements etc
   - Doesn't integrate top-down and bottom-up design
   - Assumes *HW is given*

3. Sketch the spiral model of system development.
   - System feasibility - spec - prototype - initial sys - enhanced sys
   - Successive refinement of system
   - Provides bottom-up feedback from previous stages

- Working through stages may take too much time

4. What is the successive refinement model ?
   - Start with mockups, move through simple systems to full-scale systems
   - Initial -> refined system

5. Sketch a hardware/software co-design flow.
   - Must architect HW & SW together: provide sufficient resources, avoid SW bottlenecks
   - Can build pieces somewhat independently, but integration is major step
   - Also requires bottom-up feedback

6. Give an example of a hierarchical design flow.
   - Embedded systems must be designed across multiple levels of abstraction: System architecture, HW/SW systems, HW/SW components
   - Often need design flows within design flows
   - [P75-76]

7. Why do companies introduce “concurrent engineering” ?
   - Large projects use many people from multiple disciplines
   - Work on several tasks at once to reduce design time
   - Feedback between tasks helps improve quality reduce number of later design problems
   - Techniques: cross-functional teams, concurrent product realization, incremental infomation sharing, integrated project management, supplier involvement, customer focus

8. What makes up a good requirements list ?
   - Correct, unambiguous, complete, verifiable, consistent, modifiable, traceable

9. Why are there different specification styles?
   - Capture functional & non-functional properties: verify correctness of spec, compare spec to implementation
   - Styles: control-oriented vs. data-oriented, textual vs. graphical
   - UML: one specification/design language

10. What is the purpose of the SDL language ?
    - Specification & Description Language
    - Used in telecommunications protocol disign
    - Event-oriented state machine model

11. Why are finite state machines insufficient for system specification ?

- Timing annotation
- Composite states: `OR` , `AND`

12. What is the purpose of composite (AND/OR) states in Statecharts ?
    - Reduce the size of the state trasition graph
    - Example Figure: [P90]

13. Explain the *CRC* card methodology.
    - **Classes**, **Responsibilities** for each classes, **Collaborators** are other classes that work with a class
    - Informal team-oriented methodology
    - Develop an initial list of classes: simple description is OK, team members should discuss their choices
    - Write initial responsibilities/collaborators: helps to define the classes
    - Create some usage scenarios: major uses of system and classes
    - Walk through scenarios: see what works and doesn't work
    - Refine the C R C
    - Add class relationships: superclass, subclass
    - Elevator example: [P100-106]

§3 Instruction Sets [22-36][107-159]

1. What characterizes a von Neumann computer architecture ?
    - **CPU + Memory**
    - Memory holds data, instructions
    - CPU fetches instructions from memory. Seperate CPU and memory distinguiashes programmable computer
    - CPU registers: program counter(PC), instruction register(IR), general-purpose registers, etc.

2. What is a Harvard architecture and why is this architecture preferred in DSP processors ?
    - [P111]
    - Can't use self-modifying code
    - Allows two simulaneous memory fetches
    - For streaming data: greater memory bandwidth, more predictable bandwidth

3. What are major differences between CISC and RISC processors ?

- Complex ISC: many addressing modes, many operations
- Reduced ISC: load/store, pipelined instruction execution

4. Why do we need a **programming model** of a CPU ?
   - ISA and registers visible to the programmer (some not: IR)

5. What is the purpose of pseudo assembly instructions or assembler directives ?
   - Some assembler directives don't correspond directly to instructions: difine current add, reserve storage, constants

6. What are the main characteristics of the SHARC DSP ?
   - 32/40-Bit IEEE **Floating-Point** Math
   - 32-Bit **Fixed-Point** MACs with 64-Bit Product & 80-Bit accumulation
   - **Circular Buffer** Add supported in HW
   - 32 Add pointers support 32 curcular Buffers
   - Six nested levels of **Zero-Overhead Looping** in HW
   - Rich, algebrac assembly language syntax
   - IS supports **Conditional Arithmetic**, Bit manipulation, divide & square root, bit field deposit and extract

7. What is a modified Harvard architecture ?
   - SHARC microarchitecture
   - Program memory can be used to store some data
   - Register file connects to: Multiplier; Shifter; ALU

8. How can a C programmer access multiple memories ?
   - DM and PM in paralell
   - `R2=DM(I1,M5), R1=PM(I8,M13);`

9. What is circular memory addressing ?
   - New data constantly arrives, each datum has a limited lifetime
   - Circular buffer to hold the data stream

10. What is the purpose of conditional instructions ?
    - Instructions may be executed conditionally
    - Conditions e.g. come from Arithmetic Status (ASTAT), Mode Control (MODE1), Loop Counter

11. Describe different variants of implementing C IF-statements by machine instructions.
    - Normal and fancy

- [148_42]

12. Which types of branches are typically supported by a processor ?
    - CALL, JUMP, RTS, RTI
    - Can be conditional
    - Add can be direct, indirect, PC-relative
    - Can be delayed or non-delayed

13. What is a delayed jump/branch ?
    - NAN

14. Why do most DSPs support zero overhead loops and how do they work ?
    - DO UNTIL instruction provides efficient looping
    - `LCNTR=30, DO label UNTIL LCE;`
    - Loop counter, label, loop counter expired
    - Loop length, last instruction in loop, Termination condition

15. How many instruction cycles does a typical DSP processor need for FIR filter computations ?
    - [154_48]

§4 Microprocessors [37-68][160-254]

1. How can CPUs support I/O ?
    - Typical digital interface to CPU [Fig.161]
    - Two types of instructions can support I/O: special-purpose I/O instructions; memory-mapped load/store instructions
    - Intel x86 provides `in, out` instructions
    - Most other CPUs use memory-mapped I/O
    - ARM(`LDR, STR`), SHARC(`DM`), High-level(`peek, poke`) examples [164-166]

2. What is “busy wait” I/O ?
    - Simplest way to program device: use instructions to test when device is ready
    - Simutaneous busy/wait I/O: `while(TRUE) { ... while(peek(IN_STATUS)==0)..}`
    - Very inefficient: CPU can't do other work while testing device; hard to do simultaneous I/O

3. How are interrupts processed ?

- Allow a device to change the flow of control in the CPU: causes subroutine cal to handle device request
- **Physical Interface**: CPU & device are connected by CPU bus; CPU & device handshake, device asserts interrupt request, CPU asserts interrupt acknowledge when is can handle the interrupt [Fig.171]
- Behavior: based on subroutine call mechanism; interrupt forces next instruction to be a subroutine call to a predetermined location
- Examples: character I/O handlers, interrupt-driven, buffer-based [174]

4. Why should interrupts be prioritised ?
    - Determine what interrupt gets CPU first
    - **Masking**: interrupt with priority lower than current prio. is not recognized until pending interrupt is complete

5. What is an interrupt vector ?
    - Determin what code is called for each type of interrupt
    - Allow different devices to be handled by different code
    - Interrupt vector table

6. What is an *NMI* ?
    - Non-Marskable Interrupt: highest priority, never masked
    - Often used for *power-down*

7. Do interrupts incur any overhead ?
    - Handler execution time
    - Interrupt mechanism overhead
    - Register save/store
    - Pipeline-related penalties
    - Cache-related penalties
    - **Interrupt sequence** [185]
    - Example SHARC interrupt [188-190]

8. What is an exception ?
    - Internally detected error: e.g. devide-by-0, illegal address
    - **Synchronous** with instruction but **unpredictable**
    - Build exception mechanism on top of interrupt mechanism
    - Are usually *prioritized* and *vectorized*

9. Give examples for traps or software interrupts.

- Trap (*software interrupt*): an exception generated by an instruction: Call OS service routine; Call supervisor mode
- ARM used SWI instruction for traps
- SHARC offers three levels of software interrupts: Called by setting bits in IRPTL register

10. What is the purpose of a co-processor ?
    - **Added function unit** that is called by instruction
    - Floating-point units are often strctured as co-processors e.g. in earlier Intel x_86 machines
    - Use **trap** if co-processor instruction received, but no co-processor attached
    - ARM allows up to 16 designer-selected co-processors

11. Describe a typical system memory hierarchy.
    - Expoitation of spatioal and temporal locality
    - The closer to CPU, the *faster, more expensive, smaller*
    - **Chip**[CPU(register file), L1 Cache], **Board or multi-chip module**[L2 Cache], Main memory, hard disk, tape, nettwork ...
    - [Fig.196]

12. What are pros and cons of caches in DSP systems ?
    - Many main memory locations are mapped onto one cache entry
    - May have caches for instructions, data, data+instructions (unified)
    - Memory access time is no longer deterministic
    - **Performance benefits**: keep *frequently-accessed* locations in fast caches, cache retrieves *more than one word* at a time, sequential accesses are faster *after first access*

13. What is a “working set” ?
    - Set of locations used by program in a given time interval

14. Which three types of cache misses exist ?
    - *Compulsory(cold)*: location has never been accessed
    - *Capacity*: working set is too large
    - *Conflict*: multiple locations in working set map to same cache entry

15. Derive the average memory access time in a two-level cache system.
    - $t_{av} = h_1 t_{L1} + (h_2 - h_1) t_{L2} + (1 - h_2) t_{main}$

16. Why do we need a replacement policy and what is LRU ?
    - Strategy for choosing which cache entry to overwrite to make room for a new memory location

- Random or Least-Recently-Used (LRU)

17. Define possible cache organizations.
    - **Fully-associative**: any memory location can be stored anywhere in the cache (almost never implemented)
    - **Direct-mapped**: each memory location maps onto exactly one cache entry
    - **N-way set-associative**: each memory location can go into one of N sets

18. What is the advantage of set-associative versus direct mapped caches ?
    - In general significantly different contents than direct mapped cache () larger size)

19. Describe the access procedure for a two-way set-associative cache.
    - Link wiki

20. Explain different cache write strategies.
    - Write-through: immediately copy write to main memory
    - Write-back: write to main memory only when location is removed from cache

21. How does instruction pipelining work ?
    - Several instructions are executed simultaneously at different stages of completion

22. Sketch a typical pipeline architecture.
    - [27] Several instructions are executed simultaneously at different stages of completion: instruction *fetch* / instruction *decode* / *execute* / memory access / write back. *ARM, SHARC*
    - [28] Five-stage pipeline architecture

23. Define the terms “latency” and “throughput” for pipelined processors.
    - [31]
    - *__Latency__: Time it takes for an instruction to get through the pipeline;
    - **throughput**: #instructions executed per time period

24. What is a pipeline bubble and why can it occur ?
    - [32] If every step cannot be completed in the same amount of time, pipeline stalls. Branches, memory system delays etc. -> reduce utilization(throughput), increase latency
    - [33] Figure.

25. What is a branch penalty and how can it be avoided ?

- To increase pipeline efficiency, **delayed branch mechanism** requires `n` instructions after branch **always executed** whether branch is executd or not.
- SHARC supports delayed and non-delayed branches: specified by bit in branch instruction, 2 instruction branch delay slot

26. What is a **superscalar processor** ?
    - Can execute several instructions per cycle: uses multiple pipelined data paths
    - Programs execute faster but itis harder to determine how much faster
    - Hard to use for real-time DSP applications

27. Why are power and energy concerns important for embedded systems ?
    - Heat depends on power consumption
    - Battery life depends on energy consumption
    - Power wall

28. What are sources for power consumption in a **CMOS** circuit ?
    - **Voltage**: power consumption proportionla to V^2
    - **Toggling**: more activity means more power
    - **Leakage**: basic circuit characteristics, can be eliminatted by disconnecting power

29. What are the main methods for **CPU power saving** ?
    - Reduce power supply voltage
    - Run at lower clock frequency
    - Disable function units with control signals when not in use
    - Disconnect parts from power supply when not in use

30. What is meant by *“static”* and *“dynamic”* **power management** ?
    - Static: does not depend on CPU activity: user-activated power-down mode
    - Dynamic: based on CPU activity: disabling of function units

31. What is visualized in a power state machine ?
    - Power modes, Wakeup time, Power consumption [Fig.252]

32. Why should low-power modes in CPUs be used carefully ?
    - Going into a power-down mode costs time & energy
    - Must determine if going in to mode is worthwhile
    - Can model CPU power states with power state machine

§5 Designing with Microprocessors [69-87][255-307]

1. What is an evaluation board ?
    - Designed by CPU manufacturer or others,
    - Includes CPU, Memory, some IO devices,
    - Permits execution on target hardware,
    - CPU manufacturer often gives out evaluation board netlist,
    - Can be used as starting point for your custom board design. [22]

2. What are possible components of an evaluation board ?
    - includes CPU, Memory, some IO devices

3. What is a bus protocol ?
    - Bus connects CPU to memory & devices.
    - Protocol controls communication between entities.
    - Determines who gets to use the bus at any particular time.
    - Govern length, style of communication.
    - [28]

4. Describe a simple handshake protocol.
    - Four-cycle: two wires enq & ack (enquiry-acknowledge)
    - [29,30]

5. Mention examples for popular bus protocols.
    - Timing
    - [33]

6. What problems are associated with debugging embedded systems ?
    - Target system may be hard to observe,
    - Target may be hard to control,
    - May be hard to generate realistic inputs,
    - Setup sequence may be complex.
    - [35]

7. How does a software debugger work ?
    - A monitor program residing on the target provides basic debugger functions,
    - Debugger should have a minimal footprint in memory,

- User program must be careful not to destroy debugger program,
- But debugger should be able to recover from some damage caused by user code.
- [36]

8. What is a *breakpoint* ?
   - Allows the user to stop execution, examine system state and change state.
   - Replace the breakpointed instruction with a subroutine call to the monitor program.
   - [37]

9. What is an *in-circuit emulator* ?
   - (ICE): a specially-instrumented microprocessor.
   - Allows to stop execution, examine CPU state, modify registers.
     Disadvantage: microprocessor-specific
   - [40]

10. Sketch the basic architecture of a *logic analyser*,.
    - A LA is an array of low-grade oscilloscopes, logic(0/1/x) display of samples signals.
    - [41] Figure.

11. What is the general procedure to validate software for embedded processors ?
    - Run on host system.
    - Run on target system.
    - Run in instruction-set simulator.
    - Run on cycle-accurate simulator.
    - Run in hardware/software co-simulation environment or virtual platform.
    - [43]

12. What is manufacturing testing ?
    - Goal: ensure that manufacturing produces defect-free copies of the design.
    - Can test by comparing unit being tested to the expected behaviour. But running tests is expensive.
    - Maximise confidence while minimising testing cost.
    - [44]

13. Define the terms “yield”, “fault model”, and “fault coverage”.
    - *Yield:* proportion of manufactured systems that work.
      Proper manufacturing maximises Yield,
      Proper testing accurately estimates Yield.
    - *Field return:* defective unit that leaves the factory.

- *Fault Model:* model that predicts effects of a particular type of fault.
- *Fault Coverage:* proportion of possible faults found by a set of tests.
  Having a fault model allows us to determine fault coverage.
- [46]

14. What are major differences between software and hardware testing ?
    - *Software:* no fault model. Verify implementation, not manufacturing. Simple tests work well to verify software manufacturing.
    - *Hardware:* requires manufacturing tests in addition to implementation verification.
    - [47]

15. Explain the stuck-at fault model. Is it realistic ?
    - Stuck-at-0: output of gate is always 0.
    - Stuck-at-1: output of gate is always 1.
    - [48]

16. How is the stuck-at fault model used for testing combinational logic ?
    - Every gate can be stuck-at-0, stuck-at-1.
    - Usually test for *single stuck-at-faults*: one fault at a time, multiple faults can mask each other.
    - Generate a test for a gate by controlling the gate's input & observing the gate's output through other gates.
    - [49]

17. What architectural support can be provided for sequential testing ?
    - A state machine is combinational logic + registers.
    - Sequential testing is considerably harder:
      A single stuck-at fault affects the machine on every cycle.
      Fault behaviour on one cycle can be masked by same fault on other cycles.
    - *Scan Chains:* A special register operates in two modes: normal & scan (forms an element in a shift register)
    - Reduces Sequential testing to combinational testing: Loading/unloading scan chain is slow, may use partial scan.
    - [50, 51]

18. What is meant by ATPG ?
    - *Automatic Test Pattern Generation:* produces a set of tests given the logic structure. (exponential problem)
    - Some faults may be from redundant logic.

- Timeout on a fault may mean hard-to-test or un-testable
- [52]

19. What is the purpose of boundary scan ?
    - Simplifies testing of multiple chips on a board.
    - Standard interface: JTAG (Joint Test Action Group), registers on pins can be configured as a scan chain.
    - [53]

§6 Program Design & Analysis [88-142][308-407]

> 1-6: P260

1. What is a **design pattern** ?
    - A generic description of a component tthat can be customized and used in different circumstances
    - Examples: data structures: linear lists, hash tables, etc.

2. Describe the **state machine model** of computation and sketch an example.
    - Keeps internal state as a variable, changes state based on inputs
    - [Fig.262]

3. For which type of applications are state machine models most suitable ?
    - Uses: control-dominated code; reactive systems

4. How are state machines implemented in the C language ?
    - Current state is kept in a variable
    - State table is implemented as a switch statement: cases define states, states can test inputs
    - Switch is repeatedly evaluated in a while loop
    - `while(TRUE){ switch(state){ case state1: ... }}`
    - `switch(state){ case A: if(in1 == 1) {x=a; state=B;} else {s=b, state=D}}`

5. Give an example for a data stream oriented design pattern.
    - Commonly used in signal processing: new data constantly arrive, each datum has a limited lifetime

- Use a circular buffer to hold the data stream

6. Why do compilers generally use intermediate representations (IRs) for program representation instead of source language code ?
   - Source code is not a good representation for programs: clumsy, language dependent, leaves much information implicit
   - Compilers derive IR to manipulate and optimize the program
   - [Fig.311]
7. What is **three address code** ?
   - *Assembly-like*, yet *machine-independent*
   - At most three operands per statement
   - Split complex expressions by inserting **temporary variables**

8. What is a **basic block** ?
   - Code with one entry and one exit

9. What is captured in a data flow graph ?
   - Describes the minimal ordering requirements on operations

10. Why is a **single assignment** form useful ?
    - Three-address-code <-> DFG

11. What is a CDFG ?
    - **Control-Data Flow Graph**: represents control and data
    - Uses DFG as components
    - Two types of nodes: decision; data flow

12. Give examples for capturing control flow in a CDFG.
    - if-then-else, switch/case, for loop

13. Give an overview of the code generation tool chain.
    - HLL --(compile)--> assembly --(asseble)--> object codes --(link)--> executable --(load)--> Prosessors

14. What is the purpose of an assembler ?
    - Generate binary for symbolic instructions
    - Translate labels into addresses
    - Handle pseudo-ops (data, etc.)

- Generally one-to-one translation

15. How does the assembler build symbol tables ?
    - Use program location counter (PLC) to determine address of each location
    - Scan program, keeping count of PLC
    - Addresses are generated at assembly time, not execution time
    - Efficient implementation: linear list too slow -> hash table
    - Hash function: map identifiers to table entries, which contain list of items with common hash key

16. Why does an assembler generally comprise two passes ?
    - Pass 1: collect symbolic addresses, generate symbol table
    - Pass 2: translate symbolic to real address, generate binary instructions
    - **Relative address generation**: Some label values may not be known at assembly time. Labels within the module may be kept in *relative form*. Must keep track of *external labels* - can't generate full binary for instructions that use external labels

17. What are pseudo-operations in assembly languages ?
    - Do not generate instructions:
    - **ORG** sets program location
    - **EQU** generates symbol table entry without advancing PLC
    - **Data statements** define data blocks

18. What are the tasks of a linker ?
    - Combines several object modules into a single executable module
    - Jobs: put modules in order; resolve labels across modules

19. Describe the major phases of a compiler.
    - Compilation = translation + optimization
    - HLL -(*parsing, symbol table*)-(*machine-independent optimizations*)-(*machine-dependent optimizations*)-> assembly

20. Describe a simple **CDFG-based code generation** procedure.
    - Source code is translated into intermediate form such as CDFG
    - CDFG is transformed / optimized
    - CDFG is translated into instructions with optimization decisions
    - Instructions are further optimized

21. How does a compiler generate code for procedure calls ?

- Need code to: call and return; pass parameters and results
- Parameters and returns are passed on **stack**
- Procedures with few parameters may use registers
- Frame pointer `FP` , Stack pointer `SP`
- ARM procedure linkage example [350]

22. Describe the addressing of composite data structure elements (arrays,structures) in compiled code.
    - 1D arrays: C array name points to 0th element
    - 2D arrays: Row-major layout
    - Structures: fields within structures are static offsets [354]

23. What are the most important objectives in DSP code optimisation ?
    - LOOP 😃

24. Exemplify some simplification rules a compiler might use for code optimisation.
    - Constant folding: `8+1=9`
    - Algebraic: `a*b+a*c=a*(b*c)`
    - Strength reduction: `a*2=a<<1`

25. What is *dead code* ?
    - `#define DEBUG 0; if(DEBUG) dbg(p1)`
    - Can be eliminated by analysis of control flow, constant folding

26. What is *procedure inlining* ?
    - Eliminates procedure linkage overhead: parameter passing; context saving; call/return instructions (pipeline hazards)
    - Example [357]

27. Why are loop transformations highly important for code quality ?
    - 90/10 rule: 90% of execution time spent in 10% of the code (loops)
    - Goals: reduce loop overhead; increase opportunuties for pipelining; improve memory system performance

28. What are the pros and cons of *loop unrolling* ?
    - Reduces loop overhead, enables some other optimizations

29. What is the effect of *loop fusion* and *loop distribution* ?
    - Fusion combines two loops into one

- Distribution breaks one loop into two
- Changes optimizations within loop body

30. How does *loop tiling* work ?
    - Breaks one loop into a nest of loops
    - Changes order of accesses within array
    - Changes cache behavior
    - Example [363]

31. What is meant by array padding ?
    - Add array elements to change mapping into cache

32. What is the task of a *register allocator* ?
    - Choose register to hold each varuable
    - Determine lifespan of variable in the register
    - Basic case: within basic block
    - Global: within procedure

33. Describe a technique for register allocation in a basic block.
    - [366]

34. What is the relation between register allocation and graph coloring ?
    - Build interference graph: one node per variable, one edge for each lifetime overlap
    - Background: map coloring

35. For which types of processors is instruction scheduling important ?
    - Non-pipelined machines without instruction-level parallelism do not need instruction scheduling
    - Any order of instructions that statisties data dependencies runs equally fast
    - In pipelined machines, execution time of one instruction depends on the nearby instructions: **opcode**, **operands**

36. What is represented in a **reservation table** ?
    - Relates instructions/time to CPU resources
    - Allows detection of inter-instruction resource conflicts
    - Example [370]

37. Which types of *inter-instruction dependencies* have to be taken into account during scheduling ?
    - Data dependence (Read-After-Write, RAW)

- Output dependenc (Write-After-Write, WAW)
- Anti dependence (Write-After-Read, WAR)

38. What is the idea of *software pipelining* ?
    - Schedules instructions across loop iterations
    - Reduces instruction latency in iteration i by inserting instructions from other iterations

39. Give an example for software pipelining on a DSP processor.
    - Software pipelining in SHARC [376-377]

40. Why is *instruction selection* important in *code generation* ?
    - May be several ways to implement an operation or sequence of operations
    - Represent operations as graphs, match possible instruction sequences onto graph

41. Outline the *template matching* approach to instruction selection.
    - [Fig.380]

42. Compare *compilers* to *interpreters* and *JIT compilers*.
    - **Interpreter**: translates and executes program statements on-the-fly
    - **Just-in-Time compiler**: compiles small sections of code into instructions during program execution. E.g. Java Virtual Machine.
    - Add performance / memory overhead

43. What makes *static prediction* of program *execution time* hard ?
    - Depend on: input data values; instruction/operand types; cache effects; pipelining effects

44. How can program *execution time* be measured ?
    - **CPU simulator**: I/O may be hard; may not be totally accurate
    - **Hardware timer**: Requires board, intrumented program
    - **Logic analyzer**: Limited logic analyzer memory depth

45. Why should best, worst, and average execution times be analysed ?
    - **Average-case**: for typical data values, whatever they are
    - **Worst-case**: for any possible input set
    - **Best-case**: for any possible input set
    - Too-fast programs may cause critical races at system level

46. What is a program path ?

- [389]

47. Which data structure helps to determine the execution times of program paths ?
    - LOOP 😃

48. Describe *“code motion”* as a *loop optimisation*.
    - [392]
    - `for(i=0; i<N*M; i++) -> i=0, X=N*M; if(i<X)...`

49. How does *induction variable elimination* work ?
    - [393]
    - Rather than recompute i*M+j for each array in each iteration, share induction variable between arrays, increment at end of loop body

50. Why are both power and energy optimisation important for DSP systems ?
    - **Energy**: ability to do work: most important in battery-powered systems
    - **Power**: energy per unit time: important even in wall-plug systems: power becomes heat

51. How can the energy consumption of a program be measured ?
    - Execute a small loop, measure current

52. What types of CPU instructions tend to consume most energy ?
    - Memory

53. Mention some general recipes for minimizing *program energy consumption*.
    - **Cache behavior**: cache too small: program burns too much energy on external memory accesses; cach too large: cache itself burns too much energy
    - First-order optimization: high performance = low energy
    - Most instructions do not show significant variances in power consumption
    - Use registers efficiently
    - Identify and eliminate cache conflicts
    - Moderate loop unrolling eliminates some loop overhead instructions
    - Eliminate pipeline stalls
    - Inlining procedures may help: reduces linkage overhead, but may increase cache conflicts

54. What are techniques for *program size optimisation* ?
    - Reduce HW cost of memory and power consumption of memory units
    - **Reduce data size**: reuse constants, variables, data buffers in different parts of code: requires careful verification of correctness

- Generate data using instructions
- **Reduce code size**: Avoid function inlining, choose CPU with compact instructions, use specialized instructions where possible

55. What are pros and cons of code compression ?
    - Use statistical compression after linking to reduce code size
    - Decompress on-the-fly

§7 VLSI Implementation [143-153][408-441]

1. What are advantages of DSP system implementation in VLSI technology ?
    - Very Large Scale Integration: $10^6 \ > 10^9$ transistors per chip
    - CMOS size min: 65/45/32/14/.. nm
    - Integration improves design: lower parasitics = higher speed; lower power; physically smaller
    - Reduces manufacturing cost, (almost) no manual assembly

2. What is described by Moore´s Law ?
    - Number of transistors per chip would grow exponentially (double every 18 mo)

3. How many transistors can be integrated on today´s VLSI systems ?
    - $> 10^9$ (Apple A12: 10^{10})

4. What are the major cost factors for integrated circuits ?
    - Large-volume IC: Packaging > Testing > ..
    - Low-volume IC: Design(Non-Recurring Engineering, NRE) > Manufacturing
    - Mask set: 1 Mio$; NRE: > 10 Mio$

5. Sketch the **Y-chart** of the VLSI design process.
    - Figure [423/15]

6. Describe the concept of hierarchical VLSI design.
    - Interior view of a component: components and wires that make it up
    - Exterior view of a component: type: body, pins
    - Example [425/18-21]

7. Exemplify the different abstraction levels in VLSI design.
    - Specification: function, cost, etc.
    - Architecture: large blocks
    - Register transfer(RTL): register files, ALUs, MUXes, buses
    - Logic: gates+registers
    - Circuits: transistor sizes for speed, power
    - Layout: determines parasitics

8. What is meant by “digital abstraction” of a circuit ?
    - Figure [435/28]

9. Compare top-down vs. bottom-up design methodologies.
    - *Top-down* design adds functional detail. Create lower levels of abstraction from upper levels
    - *Bottom-up* design creates abstractions from low-level behavior
    - Good design needs both top-down and bottom-up efforts

10. What is *“back annotation”* ?
    - Copy performance numbers to earlier stages

11. What is the difference between design validation and manufacturing test ?
    - **Design validation:**
    - Must check at every step that errors haven't been introduced
    - The longer an error remains, the more expensive it becomes to remove it
    - Forward checking: compare results of less- and more-abstract stages
    - Back annotation: copy performance numbers to earlier stages
    - **Manufacturing test:**
    - Not the same as design validation: just because the design is right doesn't mean that every chip coming off the line will be right
    - Must quickly check whether manufacturing defects destroy function of chip

§8 RTL Components [154-159][442-460]

1. What is the functionality of a *barrel shifter* ?
    - Can perform n-bit shifts in a single cycle
    - Efficient layout

- Does require transmission gates and long wire

2. Sketch the architecture of a *barrel shifter*.
   - Accepts 2n data inputs and n control signals
   - Producing n data outputs
   - 2D array of 2n vertical times n horizontal cells
   - Input data travels diagonally upward. Output wires travel horizontally
   - Control signals run vertically. Exactly one control signal is set to 1, turning on all transimission gates in that column
   - Large number of cells, but each one is small. Delay is large, considering long wires and transmission gates
   - Figure [449/42-43]

3. What is the functionality of a full adder ?
   - One bit sum
   - $s_i = a_i \oplus b_i \oplus c_i, c_{i+1} = a_i b_i + a_i c_i + b_i c_i$

4. What is the critical path in a *ripple-carry adder* ?
   - N-bit adder built from FA
   - Delay of RCA goes through all carry bits
   - Figure [453/46]

5. Describe a *carry-lookahead adder*.
   - First compute carry propagate, generate: $P_i = a_i + b_i, G_i = a_i b_i$
   - Compute sum and carry from P and G: $s_i = c_i \oplus P_i \oplus G_i, c_{i+1} = G_i + P_i c_i$
   - Recursively expand carry formula:
   - $c_{i+1} = G_i + P_i(G_{i-1} + P_{i-1} c_{i-1})$
   - $c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1}(G_{i-2} + P_{i-1} c_{i-2})...$
   - Expanded formula does not depend on intermediate carries
   - Allows carry for each bit to be computed independently
   - Deepest carry expansion requires gates with large fanin: large, slow
   - CLA unit requires complex wiring between adders and lookahead unit
   - Values must be routed back from lookahead unit to adder

6. Sketch a *bit-serial adder*.
   - May be used in signal-processing arithmetic where fast computation/throughput is important but latency is unimportant
   - Figure [459/52]

§9.1 Architecture and Chip Design [160-180][461-517]

1. What is a **register-transfer (RT)** system ?
    - A sequential machine (Finite state machine)
    - RT design is **structual** - complex combinations of state machines may not be easily described solely by a large state trasition graph
    - RF design concentrates on **functionality** not details of logic design

2. Why is it useful to separate a system into **data path** and **controller** ?
    - Good way to structure a system
    - Data path executes regular operations (arithmetic, etc.), holds registers with data-oriented state
    - Controller evaluates irregular functions, sets control signals for data path

3. How are conditionals modelled by RT components ?
    - *Multiplexers* are implied by *conditionals*. Must evaluate multiple paths to determine which sources of data for registers
    - Multiplexers also come from sharing ALUs and registers

4. What is an ASM chart ?
    - *Algorithmic State Machine Chart* is a register-transfer description
    - Describe functions without choosing a partitioning between control & data
    - Once specified function, can refine it into block diagram which partitions data and control

5. What is meant by “function unit sharing” ?
    - Function units implementing data operations may ba reused across states, but *multiplexers* will be requied to route values to the shared function units.
    - Example [481/22]

6. What is the relation between ASMs and Moore/Mealy automata ?
    - ASM Chart describes a **Moore** sequential machine
    - If the logic associated with an ASM chart fragment doesn't correspond to a legal sequential machine, then it isn't a legal ASM chart
    - Conditional can evaluate only present state or primary input value on present cycle

7. How do ASMs help in *data path* and *controller partitioning* ?
    - ASM chart notation helps identify data, control

- Once choose what values and operations go into the data path, can determine by elimination what goes into the controller
- Structure of the ASM chart gives structure of controller state transition graph

8. What is the purpose of *hardware description languages (HDLs)* ?
   - Textual languages for describing HW: *structure*, *timing*, *(bit accurate) function*
   - Most people today use *textual* languages rather than *schematics* for most digital designs: schematics make poor use of screen space
   - VHDL, Verilog

9. Define *(1)"compiled code simulation"* and *(2)"event-driven simulation"*.
   - (1) Generate program that evaluates a HW block
   - (2) Propagate events through simulation, don't simulate a block until its inputs change
   - (2) Example [492/33]

10. Describe the use of the *timewheel* data structure in simulation.
    - Timewheel is a data structure in the simulator that efficiently determines the order of events processed
    - Events are placed on the timewheel in time order
    - Events are taken out of the head of the timewheel to process them in order
    - Example [495/36]

11. Give examples for *structural* and *behavioural* hardware models.
    - *structural* modeling describes the connections between components. Netlists or schematics are structural models
    - *behavioural* models describes the functional relationship between inputs and outputs, similar to programming but values are events

12. What is a *testbench* ?
    - Model used to exercise a simulation: provides stimulus, checks outputs
    - Help automate design verification: Rerun edited module against testbench; Run models at behavioral, RTL levels against the same testbench

13. What is meant by a *"synthesis subset"* of an HDL ?
    - Synthesizable, produces consistent simulation results

14. What is *RT synthesis* ?
    - Register-transfer synthesis: most common type of synthesis

- Synthesizes gates from abstract RT model: registers are explicit, some tools will infer storage elements (be careful)
- Optimized for performance, area, power

15. Define the term *high-level synthesis* (or *behavioral synthesis*).
    - Constructs RT model from partially untimed behavioral specification
    - `x <= a+b; y <= c+d;`
    - `if(z>0) then w <= e+f; end if;`
    - How many clock cycles are required? -> Schedule operations

16. What is *ASAP/ALAP* scheduling ?
    - As-soon-as-possible / As-late-as-possible scheduling
    - Examples [505/45]

17. What is the relation between ASAP/ALAP and critical paths in data flow graphs ?
    - ASAP Time = ALAP Time
    - NAN

18. What is the impact of operation chaining on the cycle time of a design ?
    - Perform series of operations in one cycle
    - Time reduction

19. Give an example of *non-additive* delay in operation chaining.
    - Ripple-Carry Adder (RCA)
    - Critical path goes through all full adders of first RCA and MSB full adder of second RCA
    - Cycle time(chained) =~ cycle time(non-chained) for RCA and large word length

20. Describe the major steps of high-level synthesis.
    - Convert behaviorla spec into single assignment form
    - Data flow graph construction
    - DFG scheduling (arranging operations in cycles or control steps)
    - Register insertion
    - Multiplexer insertion
    - Controller generation
    - High-level synthesis result

§9.2 Architecture and Chip Design [181-205][518-568]

1. What are the advantages of hardware description languages (HDLs) ?
    - The major benefit of the language is fast design and better verification. The Top-down design and hierarchical design method allows the design time; design cost and design errors to be reduced. Another major advantage is related to complex designs, which can be managed and verified easily.

2. What is the simulation semantics of VHDL processes ?
    - Event-driven simulation:
    - Propagate events through simulation
    - Don't simulate a block until its inputs change
    - Example [492/33-34]

3. What is an entity in VHDL ?
    - An entity is a black box with declaration of inputs and outputs.

4. Why can a VHDL entity have multiple architectures ?
    - An architecture statement defines the structure or description of a design and is bounded with an entity. VHDL allows an entity to have multiple architectures. Since architecture describes what is inside an entity, it can be written in different ways by different designers.

5. Are there differences in the use of HDLs like VHDL or Verilog for synthesis and simulation purposes ?
    - VHDL and Verilog are designed for simulation
    - *Synthesis Subset*: synthesizable, produces consistent simulation results
    - Different tools may use different synthesis subsets

6. How does a VHDL testbench work ?
    - Model used to exercise a simulation: provides stimulus, checks outputs
    - Help automate design verification: Rerun edited module against testbench; Run models at behavioral, RTL levels against the same testbench

7. Sketch a sample VLSI design flow.
    - Figure [519/2]

8. What are methods to estimate speed or area of designs at early stages ?
    - Estimation techniques vary with module:

- Memories may be generated once size is known
- Data paths may be estimated from previous design
- Controllers are hard to estimate without details
- Estimates must include speed, area, power

9. What is the purpose of *floorplanning* ?
   - Early in design: prepare a floorplan to budget area, wire area/delay. Tradeoffs between blocks can be negotiated
   - Late in design: make sure the pieces fit together as planned. Implement the global layout

10. What is meant by state assignment in logic synthesis ?
    - For controllers, good state assignment usually requires EDA tools
    - Logic synthesis is an option: very good for non-critical logic; can work well for speed-critical logic
    - Logic synthesis system may be sensitive to changes in the input specification

11. What are the main application areas for *full custom layout* ?
    - Full custom layout design (very tedious) most likely for datapaths, least likely for random logic off critical path

12. What is covered by *design validation* ?
    - Layout (design rule check = DRC)
    - Circuit performance
    - Clock distribution
    - Functionality
    - Power consumption

13. What is meant by *tapeout* ?
    - *Tapeout*: Generating final files for masks. Shipped to mask-making house
    - *Pre-tapeout verification*: is important since it will take months to get results from fab.
    - *Tapeout party* follows. Size of party depends on importance of chip design project.

14. Sketch a basic CPU data path.
    - Figures [539/22-23]

15. What influences design decisions about the number of *register file ports* ?
    - Register file is an SRAM (Static Random-Access Memory)
    - Additional ports add area, increase access time
    - But additional ports also reduce number of cycles required for an operation

- SRAM delay grows linearly in number of ports
- Area and power consumption grow quadratically with added ports
- At least two ports makes sense for data path throughput

16. What is a wiring plan ?
    - Data path wiring plan. Figure [545/28]
    - Cannot ignore wiring during block placement: large wiring areas may force rearrangement of blocks
    - Wiring plan must consider area and delay of critical signals
    - Blocks divide wiring area into routing channels

17. What is a bricks-and-mortar floorplan ?
    - Figure [548/31]

18. What is the difference between channel and switchbox routing ?
    - *Channel routing*: Channel may grow in one dimension to accommodate wires; Pins generally on only two sides
    - *Switchbox routing*: Cannot grow in any dimension; Pins are on all four sides, fixing dimensions of the box

19. Is the definition of routing channels unique ?
    - Channels end at block boundaries. Several alternate channel definitions are possible
    - Changing spacing changes relationship between block edges
    - Example [556/39]

20. What is a *channel graph* and what is it used for ?
    - Nodes are channels, edges placed between two channels that touch
    - Shows paths between channels
    - Can be used to guide global routing
    - Example [557/40]

21. What is a “windmill” floorplan ?
    - Channels must be routed in order: wire out of end one channels creates pin on side of next channel.
    - Can create an unroutable combination of channels with circular constraints -> windmill
    - Example [560/43]

22. When is a floorplan called *“sliceable”* ?
    - Can be recursively cut in two without cutting any blocks

- Guarantee to have no windmills, therefore guarantee to have a feasible order of routing for the channels
- Slicability is a desirable property for floorplan

23. What is *global routing* ?
    - Goal: assign wires to paths through channels
    - Don't worry about exact routing of wires within channels
    - Can estimate channel height from global routing using congestion

24. How does *line probe routing* work ?
    - Heuristic method for finding a short route
    - Works with arbitrary combination of obstacles
    - Does not explre all possible paths -> not optimal

25. Why is *switchbox* routing more difficult than channel routing ?
    - Can't expand a switchbox to make room for more wiring
    - Switchbox may be defined by intersection of channels
    - Frequently need more experimentation with wiring order because nets may block other nets

§10.1 EDA Systems and Algorithms [206-224][569-612]

1. What is the purpose of *interchange languages* in CAD systems ?
    - Tools aren't very useful if they don't talk to each other
    - SystemC (IP blocks); VHDL, Verilog (function and structure);
    - EDIF (netlists); GDS, CIF (masks)

2. What is the difference between a *database* and a *translator based* approach to design interchange ?
    - Figure [574/5]
    - Database: hub-and-spoke
    - Translator: Nx(N-1) tools

3. What is *back annotation* ?
    - Often want to iteratively improve design
    - Back Annotation updates a more-abstract design with information from later design stages.
    - Example: annotate logic schematic with extracted parasitic Rs and Cs

- Back Annotation requires tools to know more about each other

4. What are the *major phases* of *layout synthesis* ?
   - **Placement** of components on the chip
   - **Routing** of wires between components
   - Placement and routing interact, but separating layout design into phases helps us understand the problem and find good solutions

5. How can you estimate the placement quality before actual wiring ?
   - Quality metrics for layout: **area**, **delay**
   - Area and delay determined in part by wiring
   - So *estimate wire length* without actually performing routing

6. What are the Euclidian and Manhattan wire length measures ?
   - Euclidean: sqrt(x^2 + y^2)
   - Manhattan: x+y

7. What is meant by “placement by partitioning” ?
   - Works well for components of fairly uniform size
   - Partition netlist to minimize total wire length using `min-cut` criterion
   - Partitioning may be interpreted as 1D or 2D layout

8. Describe the idea of **min-cut bisecting** partitioning.
   - Figure [582/14]
   - Swap A and B:
   - B drags 1 net; A drags 3 nets. Total cut increase: 4
   - Probably not a good swap, but must be compared with other paris and other subsequent moves

9. What is the main advantage of the *Kernighan-Lin* partitioning algorithm ?
   - Cumpute min-cut criterion: count total net cut change
   - Algorithm exchanges sets of nodes to perform hill-climbing: finding improvements where no single swap will improve the cut
   - Recursively subdivide to determine placement detail

10. What is the difference between *global* and *detailed routing* ?
    - Global routing assigns nets to routing areas
    - Detailed routing designs the routing areas

11. Why is *net ordering* important ?
    - Net ordering is a major problem. Order in which nets are routed determines quality for results
    - Net ordering is heuristic

12. Why are *Steiner trees* of interest in routing ?
    - A Steiner point is an intermediate point for the creation of new branches
    - Finding optimal Steiner points is difficult
    - Steiner tree problem <-> minimum spanning tree problem / shortest path

13. Describe *Lee´s maze routing algorithm*.
    - *Initialisation*: Select start point, mark with 0; i=0
    - *Wave expansion*: REPEAT mark all unlabeled neighbors of points marked with i with i+1, i=i+1; UNTIL target reached OR all points marked
    - *Backtrace*: Return from target to start point along descending marks
    - *Clerance*: Block the path for future wirings, delete all marks

14. Where does the channel routing problem occur ?
    - Routing channels: Tracks from a grid for routing: Spacing between tracks is center-to-center distance between wires; Track spacing depends on wire layer used.
    - Different layers are generally used for horizonal and vertical wires: can be routed relatively independently
    - Placement of cells determines placement of pins
    - Pin placement determines difficulty of routing problem
    - *Density*: lower bound on number of horizonal tracks needed to route the channle: maximum number of nets crossing any vertical cut

15. Describe the *left-edge channel routing algorithm* and its limitations.
    - Basic channel routing algorithm
    - Assumes one horizonal segment per net
    - Sweep pins from left to right: assign horizontal segment to lowest available track
    - *Liminations*: Some combinations of nets require more than one horizontal segment per net

16. What is a *“dogleg”* in channel routing ?
    - Has more than one horizontal segment
    - Figure [603/35]

17. What are the main ideas of the *Rivest-Fiduccia* channel router ?
    - Routes from left to right. Assign all nets that cross the current column to tracks

- Heuristics: [606/37]
- Make connections to pins;
- Add jogs to put multitrack net into one track
- Add jogs to reduce distance in multitrack nets
- Add jogs to move net toward next pin
- Add tracks when necessary

18. Describe the *scan line algorithm* for Boolean mask operations. What is it used for ?
    - Mark each edge of polygon with direction
    - Sweep scan line across layout
    - At each point on scan line, count number of left-hand and right-hand edges to determine what rectangle that point is in
    - Example [611/43]

§10.2 EDA Systems and Algorithms [225-238][613-653]

1. What is a Boolean function and how can it be represented ?
    - $f : 0, 1^n - > 0, 1$ e.g. f=a'b+ab'
    - a: *variable*, a and a': *literals*, ab': *term*
    - Irredundant, if no *literal* can be removed without changing the function's truth value

2. What is a “complete” set of functions ?
    - A set of functions $F = f1, f2, ...$ is complete if every Boolean function can be generated by a combination of the functions in F.
    - NAND is a complete set. NOR is a complete set. {AND, OR} is not complete
    - Not complete -> cannot design arbitrary logic

3. What tasks are involved in **logic synthesis** ?
    - Goal: create a logic gate network which performs a given function or set of functions
    - Input is Boolean formulas, gates also implement Boolean functions
    - **Logic synthesis**: maps onto available gates, restructures for delay, area, testability, power, etc.
    - Phases:
    - *Technology-independent* optimisations work on logic representations that do not directly model logic gates

- *Technology-dependent* optimizations work on the available set of logic gates

4. Why is library binding required ?
    - Transformation from technology-independent to technology-dependent is called *library binding*, or technology mapping
    - Rewrites Boolean network in terms of available logic functions
    - Can optimise for both area and delay
    - Can be viewed as a pattern matching problem, tries tto find pattern match which minimizes area/delay cost

5. What are methods for technology-independent logic optimisation ?
    - Don't know exact gate structure, but can estimate final network cost:
    - *Area* estimated by number of liberals (true or complement forms of variables)
    - *Delay* estimated ba path length (logic depth)

6. Define the terms *“cover”* and *“cube”*.
    - Each way to write a function as a sum-of-products is a *cover* since it covers the on-set.
    - A cover is composed of *cubes*: product terms which define a subspace cube in the function space

7. How can covers be minimized ?
    - Espresso optimisation loop:
    - *Expande*: make cubes as large as possible without changin on-set (except don't-cares)
    - *Make irredundant*: remove cubes already covered by other cubes
    - *Reduce*: make cubes as short as possible
    - Example [627/16]

8. How can *don´t cares* be exploited during logic optimisation ?
    - Partially-specified functions:
    - Don't-cares can be implemented in either on-set or off-set
    - Don't-cares provide the greatest opportunities for minimization in many cases

9. Describe a *technology mapping* procedure.
    - Write Boolean network in canonical NAND form
    - Write each library gate in canonical NAND form. Assign cost to each library gate
    - If network is a tree, can use *dynamic programming* to select minimum-cost cover of network by library gates.
    - Try to cover tree from primary inputs to primary outputs
    - Proceed one gate at a time. At the next level, select minimum-cost cover at that point

- Latest selection may require removing some earlier matches
- Example [632/21]

10. What is a *fault model* used for ?
    - Possible location of faults; I/O behavior produced by the fault
    - Good news: if we have a fault model, we can test the nettwork for every possible instantiation of that type of fault
    - Bad news: it is difficult to enumerate all types of manufacturing faults

11. Describe the *stuck-at* fault model.
    - Stuck_at_0/1: logic gate output is always stuck at 0 or 1, independent of input values
    - Correspondence to real manufacturing defects depends on logic family
    - Experiments show that 100% stuck-at-0/1 fault coverage corresponds to high overall fault coverage
    - Test: set gate inputs, observe gate output, compare fault-free and observed gate output
    - Test vector: set of gate inputs applied to a system

12. What is the general procedure for combinational logic testing ?
    - **Controlling** the inputs of possibly (interior) gates
    - **observing** the outputs of possibly (interior) gates
    - Example [644/33-38]

13. Describe the main ideas of the *PODEM algorithm* for fault propagation.
    - Goal: propagate D value to primary outputs
    - PODEM performs combinational test generation
    - Uses five values: 0,1,D,D',X. Start all values at X
    - In worst case, must examine all possible inputs, but can be implemented to run quickly
    - Example [650/40]