1 DSP INTEGRATED CIRCUITS

1.1 a) See sections 1.6, 1.7.3
b) See section 1.6.3
c) See chapter 1

1.2 a) A specification specifies the goals that is going to be achieved and a procedure to verify or validate that these goals has been accomplished.
Example: See CCITT H.261, page 8.

b) Only the inputs and the outputs and the function of the module is described in a behavioral description.
Examples:
Behavioral description: Transfer function of a filter, state-space representation of a control system, and amplifier.
Non-behavioral description: LC ladder filter and emitter-follower.

c)
```
function Abstraction(N:Integer): Integer;
    begin
      Abstraction := N div 2;
    end;

function noAbst(N:Integer): Integer;
    begin
      noAbst := N div a_global;
    end;
```

1.3 a) Repeated specification–synthesis steps – validation/verification steps..

b) The system is partitioned into a hierarchy of modules (abstractions). Any regularity is exploited.

1.4 ASIC: Less power consumption, chip area, higher throughput, more costly design process, longer design time, less flexibility, ….

Standard DSPs: Flexible, design errors can easily be corrected, easy to update as better DSPs becomes available, well-known design process, high power consumption, higher unit cost, low throughput, ….

1.6 Since, we have $\lim_{n\to\infty} \frac{ln(n)}{n^{\alpha}} = \lim_{n\to\infty} \frac{\frac{1}{n}}{\alpha\, n^{\alpha-1}} = 0$

Hence $ln(n)$ grows no faster than n^{α} for $\alpha > 0$.

1.7 We have $\lim_{n\to\infty} \frac{n^k}{a^n} = \lim_{n\to\infty} \frac{k\, n^{k-1}}{a^n\, ln(a)} = \lim_{n\to\infty} \frac{n^k}{a^n} \frac{k}{n\, ln(a)} = 0$

Hence n^k grows no faster than a^n.

1.8 In order to demonstrate the difference in complexity we compare algorithms with different complexity that are executed on a 1 MIPS computer. Notice, the comparatively slow growth of the two first algorithms while the last grows very fast.

Algorithm	$O(n)$	$O(n\ log_2(n))$	n^2	n^3
10	10 µs	33 µs	100 µs	1 ms
100	100 µs	664 µs	10 ms	1 s
1000	1 ms	9.97 ms	1 s	16.7 min
10000	10 ms	133 ms	100 s	278 h
100000	100 ms	19.93 s	2.8 h	31 710 years

1.10
```
entity Full_Adder is
   port(X, Y, Carry_in: in bit; Sum, Carry: out bit);
end Full_Adder;
architecture Behavioral_View of Full_Adder is
begin
   process
        variable n: integer;
        constant Sum_vector: bit_vector (0 to 3) := "0101";
        constant Carry_vector: bit_vector (0 to 3) := "0011";
   begin
        wait on X, Y, Carry_in;
        n :=0;
        if X = '1' then n := n + 1; end if;
        if Y = '1' then n := n + 1; end if;
        if Carry_in = '1' then n := n + 1; end if;
        Sum <= Sum_vector(n) after 3 ns;
        Carry <= Carry_vector(n) after 2 ns;
   end process;
end Behavioral_View;
architecture Data_Flow_View of Full_Adder is
signal Temp: bit;
begin
   Temp <= X or Y after 1 ns;
   Sum <= Temp or Carry_in after 2 ns;
   Carry <= (X or Y) or (Temp and Carry_in) after 1 ns;
   end process;
end Data_Flow_View;
architecture Structural_View of Full_Adder is
component Half_Adder port(A, B: in bit; S, C: out bit);
end component;
component OR_Gate port(A, B: in bit; Out: out bit);
end component;
signal Temp1, Temp2, Temp3: bit;
begin
U1:     Half_Adder port map(X, Y, Temp1, Temp2);
U2:     Half_Adder port map(Temp3, Carry_in, Temp3, Sum);
U3:     OR_Gate port map(Temp1, Temp3, Carry);
end Structural_View;
```