Question A1

a) Conditional and selection statements in combinational logic synthesis must be complete. Here is an example of a correct description:

```vhdl
always_comb
  case ({A,B})
    2'b00: Y = 1'b1;
    2'b01: Y = 1'b0;
    Default: Y = 1'bx;
  endcase
```

b) **unique** and **priority** are implicit assertions.

**unique** asserts that there is no overlap in a series of conditions, i.e., they are mutually exclusive and hence it is safe for the expressions to be evaluated in parallel.

Some compilers are able to detect potential overlap and can issue compile-time warnings. Simulators issue run-time warnings if conditions are evaluated not to be mutually exclusive. Warnings are also issued if no condition is true, or if it is possible that no condition is true.

**priority** asserts that a series of conditions shall be evaluated in the order listed. A warning is issued if no condition is true, or it is possible that no condition is true.

```vhdl
// multiplexer showing the use of unique
module mux8to1(
  input logic [7:0] i, input logic [2:0] sel, output logic outp);

always_comb // note absence of sensitivity clause
unique case (sel) // unique will force completeness test
  0: outp = i[0];
  1: outp = i[1];
  2: outp = i[2];
  3: outp = i[3];
  4: outp = i[4];
  5: outp = i[5];
  6: outp = i[6];
  7: outp = i[7];
endcase
```
Question A2

a)  
- when writing code for sequential logic, use always_ff and always_latch blocks with incomplete conditional statements  
- avoid temptation to mix complex combinational and sequential logic in one block; partition designs and use multiple blocks  
- for state machines use edge-triggered flip-flops (always_ff)  
- for general data storage use latches and RAMs (always_latch)  

b)  
// 32-byte RAM with false synchronous read  
module ram32x8 (input logic we, readclk, re, input logic [4:0] address, input logic [7:0] din, output logic [7:0] dout);  
logic [7:0] ram[31:0]; // this 2-dimensional array defines the ram memory  
// write block  
always_latch  
  if (we)  
    ram[address] <= din;  

// synchronous read block  
always_ff @(readclk)  
  if(re)  
    dout = ram[address];  
  else dout = 8'bzzzz_zzzz;  
endmodule  

c)  
testbench should include initial processes to define a read clock synchronous read, write cycles to several addresses followed by corresponding read cycles.
### Question A3

(a) If statements are intended for use in both simulation and synthesis synthesis. If simply checks if an expression is true while assert generates an error condition if an assertion fails. Assert is ignored in synthesis and is only used in simulation.

(b) Simplest implementation:

```plaintext
assert property (!((Read && Write));
```

(c) module sm (input logic clk, A, output logic Y);
logic [1:0] Q, Qnext;

```plaintext
always_ff @(posedge clk)
    Qnext <= Q; // flip-flop memory

always_comb // Qnext logic
    unique case (Q)
    2'b00: Qnext = 2'b01;
    2'b01: Qnext = A ? 2'b10 : 2'b01;
    2'b10: Qnext = 2'b00;
    default: Qnext = 2'b01;
endcase

always_comb // output logic
    unique case (Q)
    2'b00: Y = 1'b1;
    2'b01: Y = ~A;
    2'b10: Y = 1'b0;
    default: Y = 1'bx;
endcase

// assertion; NOTE: must implication: |->
assert property (@(posedge clk) (Q==2'b10) |-> ##1 (Q==2'b00)
```

endmodule