Solutions to Assignment 1

Question 1

```
case(i)
  0: // using only logic operators (&, |, ^)
    begin
       sum = a ^ b ;
       carry = a & b ;
    end
  1: // using only ternary operators (?:)
    begin
       sum = a ? ( b ? 0 : 1 ) : ( b ? 1 : 0 ) ;
       carry = a ? ( b ? 1 : 0 ) : 0 ;
    end
  2: // using only if/else statements
    if ( a )
      if ( b ) begin
         sum = 0 ;
         carry = 1 ;
      end
      else begin
         sum = 1 ;
         carry = 0 ;
      end
    else
      if ( b ) begin
         sum = 1 ;
         carry = 0 ;
      end
      else begin
         sum = 0 ;
         carry = 0 ;
      end
  3:
    // using only a case statement (and concatenation)

    //case({a,b})
    //  2'b00: begin sum = 0; carry=0 ; end
    //  2'b01: begin sum = 1; carry=0 ; end
    //  2'b10: begin sum = 1; carry=0 ; end
    //  2'b11: begin sum = 0; carry=1 ; end

    // using only one case statement and no operators
    case(a)
      b: // a == b (00,11)
        begin
           sum=0 ;
           carry=a ;
        end
      default: // {a,b} == (01,10)
        begin
           sum=1;
           carry=0;
        end
    endcase
  4: // using only concatenation ({,}), addition (+)
    // and bit subscript (slicing) ([:]) operators
    begin
       logic [1:0] s ;
       s = {1'b0,a}+{1'b0,b} ;
       {carry,sum} = s[1:0];
    end
  5: // using only initialized array(s), concatenation
    // ({,}) and the indexing operator ([])
    begin
     static logic [1:0] tab[4]='{2'b00,2'b01,2'b01,2'b10} ;
       sum = tab[{a,b}][0] ;
       carry = tab[{a,b}][1] ;
    end
endcase
```

Question 2

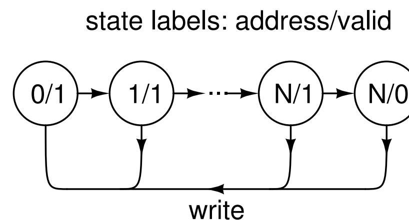
```
module mul8 ( input logic [7:0] a,
              input logic [7:0] b,
              output logic[15:0] c ) ;

     always_comb begin
        c = '0 ;
        for ( int i=$low(b) ; i<=$high(b) ; i++ )
          c += b[i] ? a << i : 0 ;
     end

endmodule
```

Question 3

The state transition table could be written as:

write	address	next address	next valid	next count
1	X	0	1	data
0	<count	address+1	1	count
0	>= count	address	0	count

The corresponding state transition diagram:



The synthesizable System Verilog description:

```
module asg1 ( input logic write,
              input logic [31:0] data,
              input logic clk,
              output logic [31:0] address,
              output logic valid ) ;
```

```
  logic [31:0] count, count_next, data_next, address_next ;
   logic valid_next ;

   always_comb begin

      address_next = address ;
      valid_next = valid ;
      count_next = count ;

      if ( write ) begin
         address_next = '0 ;
         valid_next = '1 ;
         count_next = data ;
      end else begin
         if ( address + 1 <= count ) begin
            address_next = address + 1 ;
         end else begin
            valid_next = '0 ;
         end
      end

   end

   always_ff @(posedge clk) begin
      address <= address_next ;
      valid <= valid_next ;
      count <= count_next ;
   end

endmodule
```