## SOLUTION NOTES

### ECAD 2002 Paper 4 Question 3 (SWM)

(*a*) The following things are wrong with the up/down counter module:

- The first line should be terminated with a ';'.
- The
  `output counter;`
  statement should be
  `output [7:0] counter;`
- `error` should be declared as a register.
- `{}` should be replaced by `begin` and `end`
- Currently the code is ambiguous: when `reset=up=down=1` then the counter could be set to zero, incremented or decremented all at the same time! Similarly `error` could be set or reset.
- reset should take the form to ensure that the reset input on the D flip-flops (used to implement the registers) is used:

```
always @(posedge clk or posedge reset)
  if(reset) begin
    ....reset statements....
  end else begin
    ....main body....
  end
```

- The button inputs require debouncing and resynchronising.
- The statement:
  `if(up)  counter <= counter + 1;`
  will result in counter being incremented every clock cycle whilst `up` is being pressed rather than being incremented once for each press. Similarly for `down`.
- `count` is used in the signature, but `counter` everywhere else.

(*b*) A corrected piece of code (part 1):

```
module UpDownCounter(clk, reset, up, down, counter, error);
  input  clk;            // input clock
  input  reset;          // reset
  input  up;             // from the up button
  input  down;           // from the down button
  output [7:0] counter;  // output the count value
  output error;          // error flag (1=error, 0=no error)

  reg [7:0] counter;
  reg error, last_up, last_down;
  // debounce the buttons
  wire db_up;
  wire db_down;
  debounce db0(clk, reset, up, db_up);
  debounce db1(clk, reset, down, db_down);
  always @(posedge clk or posedge reset)
    if(reset) begin  // reset all flip-flops
      counter <= 0;
      error   <= 0;
    end else begin
      // remember last state of debounced buttons
      last_up   <= db_up;
      last_down <= db_down;
      // if both buttons are pressed then indicate an error
      if(db_up && db_down) error <= 1;
      // if the up button has just been pressed, then increment the counter
      else if(!last_up && db_up)     counter <= counter + 1;
      // if the down button has just been pressed, then decrement the counter
      else if(!last_down && db_down) counter <= counter - 1;
    end
endmodule
```

Corrected code (part 2):

```
// simple module to debounce a button
module debounce(clk, reset, in, out);
  input  clk;           // input clock
  input  reset;         // reset
  input  in;            // async. input
  output out;           // async. output

  reg [7:0] samples;
  reg [19:0] delay;         // delay is a 20 bit counter
  // N.B. the size of delay needs to reflect the clock rate and the
  // mechanical periodicity of the bouncing button.  The size of delay
  // must, therefore, be modified to reflect the system requriements.
  reg out;
  always @(posedge clk or posedge reset)
    if(reset) begin         // reset all registers
      out <= 0;
      samples <= 0;
      delay <= 0;
    end else begin
      delay <= delay + 1;   // increment delay
      // when delay is zero, sample input into a shift register
      if(delay==0)
        samples <= {samples[6:0],in};

      // if bits 1 to 7 of the sample are all zero the output a zero
      // N.B. bit 0 of sample is not checked because it may be
      // metastable and we wish to allow time for metastability to resolve
      if(samples[7:1]==0) out <= 0;

      // if bits 1 to 7 of the sample are all one then output a one
      if((~samples[7:1]==0)) out <= 1;
    end
endmodule
```