DSP Project: Lock-In Amplifier

... don’t worry your’s will be smaller and simpler.
The DSP projects will follow the schedule below:

**October 8, 2007:** DSP design competition launch (this document is released).

**October 26, 2007:** Formal project proposals are due.

**October 29, 2007:** Project proposals are graded and returned.
→ Project funds are released (you should start buying ASAP so that you will have your components by the next lab).

**October 29 – November 22, 2007:** Project construction.

**November 26-30, 2007:** Project debugging.

**December 3, 2007:** Oral presentations and website launch.

**December 4-5, 2007:** Official device performance testing and review.
DSP Project Purchases

- **Budget =** $250.00 USD per team.

- All purchases will go through the instructor to Sylvia Stout.

- You must provide an internet shopping cart printout with all the part numbers. An electronic version is useful and will help speed up the purchase.

- If you buy a part at a local store, then you must keep the receipt and submit it to the instructor.
You are encouraged to use the parts in the lab to perfect your design, however your final device should include only new non-lab parts (except wire).

The course will provide Altera USB-blaster cables for programming the FPGA configuration chips in Active Serial mode.
State Machine

Definition: A machine that makes predictable transitions through a sequence of states, based on external inputs and the current state of the machine.
module StateMachine_v1(clk, reset, out);
  input clk, reset; // clock and reset declarations
  output reg out;  // output register declaration

  reg [1:0] state; // this register holds the state of the machine

  always @(state) // This always block indicates how the output
  begin // should relate to the machine state
    case (state)
      2'b00: out = 1'b0;
      2'b01: out = 1'b1;
      2'b10: out = 1'b0;
      2'b11: out = 1'b0; // only required for completeness (not necessary)
      default: out = 1'b0; // default statement guarantees completeness (not necessary)
    endcase
  end

  always @(posedge clk or posedge reset) // This always block constructs the
  begin // sequence table
    if (reset==1) state = 2'b00;
    else
      case (state)
        2'b00: state = 2'b01;
        2'b01: state = 2'b10;
        2'b10: state = 2'b00;
        2'b11: state = 2'b00; // only required for completeness (not necessary)
        default: state = 2'b00; // default statement guarantees completeness (not necessary)
      endcase
    end
  end
endmodule
State Machines in Quartus II

Tools > Netlist Viewers > State Machine Viewer

Note: The Quartus II compiler will only find the State Machine if you follow the Verilog example code fairly closely.
Shift Register Circuit

serial input

Clock trigger

D Q

bit-0 OUT

D Q

bit-1 OUT

D Q

bit-2 OUT

serial output
module ShiftRegister(serial_input, clock, reset, output_register, serial_output);

  input serial_input;  // serial input line of the shift register
  input clock;         // clock input line of the shift register
  input reset;         // reset line for resetting the shift register to ZERO
  output reg [7:0] output_register; // Register for the shift register
  output serial_output; // serial output line |equivalent to output_register[7]|

initial
  begin
    output_register = 8'b00000000; // Start with an empty shift register
  end

// connect the serial output line to the bit 8 of the shift register
assign serial_output = output_register[7];

always @(posedge clock or posedge reset)
  begin
    if (reset == 1)
      begin
        output_register <= 8'b00000000; // reset the shift register to ZERO
      end
    else
      begin
        // shifts all the shift register bits to the next bit
        output_register <= output_register << 1;

        // loads the serial input into the first shift register bit
        output_register[0] <= serial_input;
      end
  end
endmodule
module ShiftRegister(serial_input, clock, reset, output_register, serial_output);

input serial_input;  // serial input line of the shift register
input clock;        // clock input line of the shift register
input reset;        // reset line for resetting the shift register to ZERO
output reg [7:0] output_register; // Register for the shift register
output serial_output; // serial output line |equivalent to output_register[7])
initial
begin
output_register = 8'h00000000; // Start with an empty shift register
end

// connect the serial output line to the bit 8 of the shift register
assign serial_output = output_register[7];

always @(posedge clock or posedge reset)
begin
  if (reset == 1)
    begin
      output_register <= 8'h00000000; // reset the shift register to ZERO
    end
  else
    begin
      // shifts all the shift register bits to the next bit
      output_register <= output_register << 1;

      // loads the serial input into the first shift register bit
      output_register[0] <= serial_input;
    end
end
endmodule

use the left bit shift operator <<