Digital Bubble Level

Vladislav Pomogaev - 26951160

September 27, 2021

1 Code

level_fsm.sv

```
/*
Description:
  An FSM which implements the functionality of a "bubble level" or "spirit level".
  Reads single-axis accelection data from the MPU-6050 by
  interfacing with an I2C Core IP from Efinix. Wakes accelerometer up first by writing to register,
  then reads register in a loop.
  Updates the status of the 9 LED's based on the acceleration measured. Contains error LED as well.

  LED Encoding table looks like so:

   Tilt      Accel.  MSB  LSB    MSB  MSB          Led Led Led Led Led Led Led Led Led   Decimal
   Degrees   (g)     Sign             Bins         8   7   6   5   4   3   2   1   0     LED Encoding
   -8       -0.139   1   -2280  -9    1111111000   1   0   0   0   0   0   0   0   0     256
   -7       -0.122   1   -1997  -8    1111111001   1   1   0   0   0   0   0   0   0     384
   -6       -0.105   1   -1713  -7    1111111010   0   1   0   0   0   0   0   0   0     128
   -5       -0.087   1   -1428  -6    1111111011   0   1   1   0   0   0   0   0   0     192
   -4       -0.070   1   -1143  -4    1111111100   0   0   1   0   0   0   0   0   0      64
   -3       -0.052   1    -857  -3    1111111101   0   0   1   1   0   0   0   0   0      96
   -2       -0.035   1    -572  -2    1111111110   0   0   0   1   0   0   0   0   0      32
   -1       -0.017   1    -286  -1    1111111111   0   0   0   1   1   0   0   0   0      48
    0        0.000   0       0   0    0            0   0   0   0   1   0   0   0   0      16
    1        0.017   0     286   1    1            0   0   0   0   1   1   0   0   0      24
    2        0.035   0     572   2    10           0   0   0   0   0   1   0   0   0       8
    3        0.052   0     857   3    11           0   0   0   0   0   1   1   0   0      12
    4        0.070   0    1143   4    100          0   0   0   0   0   0   1   0   0       4
    5        0.087   0    1428   6    101          0   0   0   0   0   0   1   1   0       6
    6        0.105   0    1713   7    110          0   0   0   0   0   0   0   1   0       2
    7        0.122   0    1997   8    111          0   0   0   0   0   0   0   1   1       3
    8        0.139   0    2280   9    1000         0   0   0   0   0   0   0   0   1       1

  The accelerometer stores each axis measurement as 16 bit. Registers are 8 bit wide.
  We only access the MSB because we don't need the resolution.
  See page 29 of https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

Inputs:
  clk_i                    - Clock
  reset_i                  - Active high reset
  i2c_busy_i               - Logic high indicates that the I2C bus is busy
  i2c_rxak_i               - Logic low indicates that the I2C slave devicereceived and
                             acknowledged the I2C transfer
  i2c_arb_lost_i           - Logic high indicates that there is arbitration lost in the I2C transfer
  i2c_write_done_i         - Logic high indicates that I2C master write data is
                             sent and ready to accept by I2C slave device
  i2c_data_out_valid_i     - Logic high indicates that I2C master read data is valid and ready to read
  i2c_data_out_i           - Read data output from the I2C core


Outputs:
  i2c_write_o              - Write to slave strobe high
  i2c_read_o               - Read from slave strobe high
  i2c_slave_addr_o         - Address of slave in 8bit format. Add trailing zero to use 7-bit addressing.
  i2c_din_o                  - Data read from slave
  i2c_command_byte_o         - Command byte sent to slave. (Register to read from)
  i2c_num_bytes_o            - Number of bytes of data to write or read. Includes the
                               command byte (if sending one byte, i2c_num_bytes_o = 'd2)
  error_led_o                - Active high if an error has occured in state machine; lost bytes,
                               error writing to slave or reading
  led_o                      - 9 bit bus that lights up based on encoding above
*/

module level_fsm (
  input wire           clk_i,
  input wire           reset_i,

  input wire           i2c_busy_i,
  input wire           i2c_rxak_i,
  input wire           i2c_arb_lost_i,
  input wire           i2c_write_done_i,
  input wire           i2c_data_out_valid_i,
  input wire   [7:0]   i2c_data_out_i,

  output wire          i2c_write_o,
  output wire          i2c_read_o,
  output logic [7:0]   i2c_slave_addr_o,
  output logic [7:0]   i2c_din_o,
  output logic [7:0]   i2c_command_byte_o,
  output logic [7:0]   i2c_num_bytes_o,

  output wire          error_led_o,
  output logic [8:0]   led_o
);

logic signed [7:0] raw_buffer;
localparam slave_address = {7'h68, 1'b0};
localparam accel_register = 8'h3D;
localparam wake_register = 8'h6b;

typedef enum logic [6:0] {
  ENSURE_BUSY_1 =       7'b0000_000, // wait for I2C controller to not be busy
  ASSIGN_WRITE_2 =      7'b0001_000, // tell I2C controller which registers to write what to
  ASSERT_WRITE_3 =      7'b0010_001, // tell I2C controller to write
  WAIT_FOR_BUSY_4 =     7'b0011_000, // wait for controller to be busy
  WAIT_FOR_DONE_5 =     7'b0100_000, // wait for controller to finish
  VERIFY_6 =            7'b0101_000, // verify that there were no errors writing register
  ENSURE_BUSY_7 =       7'b0110_000, // verify that the controller is not busy
  ASSIGN_WRITE_8 =      7'b0111_000, // tell I2C controller which registers to read
  ASSERT_READ_9 =       7'b1000_010, // tell I2C controller to read register
  WAIT_FOR_BUSY_10 =    7'b1001_000, // wait for controller to be busy
  WAIT_FOR_VALID_11 =   7'b1010_000, // wait for controller to finish read
  VERIFY_12 =           7'b1011_000, // verify that no errors occured, register info
  LED_OPERATION_13 =    7'b1100_000, // output LED pattern
  ERROR =               7'b1101_100
} state_e;

state_e state;

// Glitch free state outputs based on last bits of current state
assign i2c_write_o = state[0];
assign i2c_read_o = state[1];
assign error_led_o = state[2];

always @(posedge clk_i) begin
  if (reset_i) begin
    state <= ENSURE_BUSY_1;
    led_o <= 9'b0;
    raw_buffer <= 8'b0;
    i2c_din_o <= 8'b0;
    i2c_command_byte_o <= 8'h0;
    i2c_slave_addr_o <= 8'b0;
    i2c_num_bytes_o <= 8'd0;
  end
  else begin
    case(state)
      ENSURE_BUSY_1: begin
        if (i2c_busy_i === 1) state <= ENSURE_BUSY_1;
```

```
          else begin
            state <= ASSIGN_WRITE_2;
            //assign DIN, command_byte, slave_addr, and num_bytes here
            //wake accelerometer by writing to wake_register 0x00
            i2c_din_o <= 8'b0;
            i2c_command_byte_o <= wake_register;
            i2c_slave_addr_o <= slave_address;
            i2c_num_bytes_o <= 8'd2;
          end
        end
        ASSIGN_WRITE_2: state <= ASSERT_WRITE_3; //assign inputs first before flashing write strobe
        ASSERT_WRITE_3: state <= WAIT_FOR_BUSY_4; //write strobe
        WAIT_FOR_BUSY_4: begin
          if (i2c_busy_i === 1) state <= WAIT_FOR_DONE_5; //wait for busy
          else state <= WAIT_FOR_BUSY_4;
        end
        WAIT_FOR_DONE_5: begin //wait for write done
          if (i2c_write_done_i === 1) state <= VERIFY_6;
          else state <= WAIT_FOR_DONE_5;
        end
        VERIFY_6: begin
          if (i2c_arb_lost_i === 0 && i2c_rxak_i === 0) state <= ENSURE_BUSY_7;
          //check that slave has acknowledged and no bits were lost
          else state <= ERROR;
        end
        ENSURE_BUSY_7: begin // wait for busy to not be high
          if (i2c_busy_i === 1) state <= ENSURE_BUSY_7;
          else begin
            state <= ASSIGN_WRITE_8;
            //assign command_byte, slave_addr, and num_bytes here
            //read the MSB of accelerometer axis measurement
            i2c_command_byte_o <= accel_register;
            i2c_slave_addr_o <= slave_address;
            i2c_num_bytes_o <= 8'd2;
          end
        end
        ASSIGN_WRITE_8: state <= ASSERT_READ_9; //wait for writing of data
        ASSERT_READ_9: state <= WAIT_FOR_BUSY_10; //flash read strobe
        WAIT_FOR_BUSY_10: begin
          if (i2c_busy_i === 1) state <= WAIT_FOR_VALID_11; //wait for busy
          else state <= WAIT_FOR_BUSY_10;
        end
        WAIT_FOR_VALID_11: begin
          if (i2c_data_out_valid_i === 1) state <= VERIFY_12; // wait for data rx
          else state <= WAIT_FOR_VALID_11;
        end
        VERIFY_12: begin
          if (i2c_arb_lost_i === 0 && i2c_rxak_i === 1) begin //check that no bits lost and no slave ack
            state <= LED_OPERATION_13;
            raw_buffer <= i2c_data_out_i; // register the data
          end
          else state <= ERROR;
        end
        LED_OPERATION_13: begin
          // enter the calculation that associates the LEDs with their respective tilt levels here
          if (raw_buffer < 8'sb11111000)
            led_o <= 9'd256;
          else if (raw_buffer < 8'sb11111001)
            led_o <= 9'd384;
          else if (raw_buffer < 8'sb11111010)
            led_o <= 9'd128;
          else if (raw_buffer < 8'sb11111011)
            led_o <= 9'd192;
          else if (raw_buffer < 8'sb11111100)
            led_o <= 9'd64;
          else if (raw_buffer < 8'sb11111101)
            led_o <= 9'd96;
          else if (raw_buffer < 8'sb11111110)
            led_o <= 9'd32;
          else if (raw_buffer < 8'sb11111111)
            led_o <= 9'd48;
          else if (raw_buffer < 8'sb00000000)
            led_o <= 9'd16;
          else if (raw_buffer < 8'sb00000001)
            led_o <= 9'd24;
          else if (raw_buffer < 8'sb00000010)
            led_o <= 9'd8;
          else if (raw_buffer < 8'sb00000011)
            led_o <= 9'd12;
          else if (raw_buffer < 8'sb00000100)
            led_o <= 9'd4;
          else if (raw_buffer < 8'sb00000101)
            led_o <= 9'd6;
          else if (raw_buffer < 8'sb00000110)
            led_o <= 9'd2;
          else if (raw_buffer < 8'sb00000111)
            led_o <= 9'd3;
          else if (raw_buffer < 8'sb00001000)
            led_o <= 9'd1;
          else
            led_o <= 9'd1;
          state <= ENSURE_BUSY_7;
        end
        ERROR: begin
          state <= ERROR;
        end
        default:  state <= ENSURE_BUSY_1;
      endcase
    end
  end

  endmodule
```

level_fsm_tb.sv

```
`timescale 1ns/100ps
/*
Description:
  An FSM testbench for the bubble level FSM. Emulates the functionality of three different states of I2C
  controller:
    A perfectly working controller
    A controller that errors on the read
    A controller that errors on the write

  Test bench tests:
    All delays
    All rx_arb errors checks
    One write state from the accelerometer to the LEDs


DUT Inputs:
  clk_i                     - Clock
  reset_i                   - Active high reset
  i2c_busy_i                - Logic high indicates that the I2C bus is busy
  i2c_rxak_i                - Logic low indicates that the I2C slave devicereceived and
                              acknowledged the I2C transfer
  i2c_arb_lost_i            - Logic high indicates that there is arbitration lost in the I2C transfer
  i2c_write_done_i          - Logic high indicates that I2C master write data is
                              sent and ready to accept by I2C slave device
  i2c_data_out_valid_i      - Logic high indicates that I2C master read data is valid and ready to read
  i2c_data_out_i            - Read data output from the I2C core


DUT Outputs:
  i2c_write_o               - Write to slave strobe high
  i2c_read_o                - Read from slave strobe high
  i2c_slave_addr_o          - Address of slave in 8bit format. Add trailing zero to use 7-bit addressing.
  i2c_din_o                 - Data read from slave
  i2c_command_byte_o        - Command byte sent to slave. (Register to read from)
  i2c_num_bytes_o           - Number of bytes of data to write or read. Includes the
                              command byte (if sending one byte, i2c_num_bytes_o = 'd2)
  error_led_o               - Active high if an error has occured in state machine; lost bytes,
                              error writing to slave or reading
  led_o                     - 9 bit bus that lights up based on encoding above
*/


module level_fsm_tb;
```

```
  logic         clk_i = 1;
  logic         reset_i;
  logic         i2c_busy_i;
  logic         i2c_rxak_i;
  logic         i2c_arb_lost_i;
  logic         i2c_write_done_i;
  logic         i2c_data_out_valid_i;
  logic [7:0]   i2c_data_out_i;

  logic         i2c_write_o;
  logic         i2c_read_o;
  logic [7:0]   i2c_slave_addr_o;
  logic [7:0]   i2c_din_o;
  logic [7:0]   i2c_command_byte_o;
  logic [7:0]   i2c_num_bytes_o;

  logic         error_led_o;
  logic [8:0]   led_o;

  level_fsm dut (
    .clk_i,
    .reset_i,
    .i2c_busy_i,
    .i2c_rxak_i,
    .i2c_arb_lost_i,
    .i2c_write_done_i,
    .i2c_data_out_valid_i,
    .i2c_data_out_i,
    .i2c_write_o,
    .i2c_read_o,
    .i2c_slave_addr_o,
    .i2c_din_o,
    .i2c_command_byte_o,
    .i2c_num_bytes_o,
    .error_led_o,
    .led_o
  );

  initial
  begin
    $dumpfile("level_fsm_tb.vcd");
    $dumpvars(0,clk_i);
    $dumpvars(1,reset_i);
    $dumpvars(2,i2c_busy_i);
    $dumpvars(4,i2c_rxak_i);
    $dumpvars(5,i2c_arb_lost_i);
    $dumpvars(6,i2c_write_done_i);
    $dumpvars(7,i2c_data_out_valid_i);
    $dumpvars(8,i2c_data_out_i);
    $dumpvars(9,i2c_write_o);
    $dumpvars(10,i2c_read_o);
    $dumpvars(11,i2c_slave_addr_o);
    $dumpvars(12,i2c_din_o);
    $dumpvars(13,i2c_command_byte_o);
    $dumpvars(14,i2c_num_bytes_o);
    $dumpvars(15,error_led_o);
    $dumpvars(16,led_o);
  end

  always #0.5 clk_i = ~clk_i;

  initial begin
    ////////////////////////////
    //////   Reset/Begin   //////
    ////////////////////////////
    // full reset with busy line
    reset_i = 1'b0;
    i2c_busy_i = 1'b1;
    i2c_rxak_i = 1'b0;
    i2c_arb_lost_i = 1'b0;
    i2c_write_done_i = 1'b0;
    i2c_data_out_valid_i = 1'b0;
    i2c_data_out_i = 8'b0;
    #1;
    reset_i = 1'b1;
    #1;
    reset_i = 1'b0;
    #1;
    ////////////////////////////
    ///   Normal operation   ////
    ////////////////////////////
    // in this section we test the normal operation of the I2C controller
    // that no errors are thrown, read, write, delays work
    // correct LED pattern shows up
    // and that we loop reading

    // make sure writing has not started
    assert(i2c_din_o == 8'b0);
    assert(i2c_command_byte_o == 8'b0);
    assert(i2c_slave_addr_o == 8'b0);
    assert(i2c_num_bytes_o == 8'b0);
    #1;
    i2c_busy_i = 1'b0; // now I2C is free
    #1;
    assert(i2c_din_o == 8'b0); // assert that correct data is going to be written
    assert(i2c_command_byte_o == 8'h6b);
    assert(i2c_slave_addr_o == {7'h68, 1'b0});
    assert(i2c_num_bytes_o == 8'd2);
    #1;
    assert(i2c_write_o == 1'b1); // assert write stobe
    #1;
    i2c_busy_i = 1'b1; // assert that we are waiting for chip to be not busy
    assert(i2c_write_o == 1'b0);
    assert(error_led_o == 1'b0);
    #1;
    i2c_busy_i = 1'b1; // assert that we are waiting for chip to be not busy
    assert(i2c_write_o == 1'b0);
    assert(error_led_o == 1'b0);
    #1;
    i2c_busy_i = 1'b0; // make sure we have not errored out yet because we have an ack and no errors
    assert(error_led_o == 1'b0);
    #1;
    assert(error_led_o == 1'b0);
    #1;
    i2c_write_done_i = 1'b1;
    #1; //wait for write to ack
    #1; //verify arb_lost and rxak low
    #1; //ensure busy is low
    assert(i2c_command_byte_o == 8'h3D);
    assert(i2c_slave_addr_o == {7'h68, 1'b0});
    assert(i2c_num_bytes_o == 8'd2);
    #1;
    assert(i2c_read_o == 1'd1);
    #1;
    i2c_busy_i = 1'b1;
    #1;
    #1;
    #1;
    i2c_busy_i = 1'b0;
    i2c_data_out_valid_i = 1'b1;
    i2c_data_out_i = 8'b11111001;
    i2c_arb_lost_i = 1'b0;
    i2c_rxak_i = 1'b1;
    //wait for valid data and busy low
    #1; //wait for busy to go low
    #1; //verify data out is valid
    #1; //verify no lost and rx did not ack, register the data
    assert(led_o == 9'd128); // assert that led output is correct
    #1; //finished test under standard conditions. since busy is low, we assert the lines again

    ////////////////////////////
    ////   Error on read    ////
    ////////////////////////////
    // in this section we confirm that upon a read error the FSM goes into error state

    assert(i2c_command_byte_o == 8'h3D);
    assert(i2c_slave_addr_o == {7'h68, 1'b0});
    assert(i2c_num_bytes_o == 8'd2);
    #1;
    assert(i2c_read_o == 1'd1); // we strobe read again
    i2c_busy_i = 1'b1;
    i2c_data_out_valid_i = 1'b0;
    #1;
    #1;// we are now waiting for busy
    #1;
    #1;
    #1;
    i2c_busy_i = 1'b0;
    #1;
    i2c_data_out_valid_i = 1'b1;
    i2c_arb_lost_i = 1'b1;
    i2c_rxak_i = 1'b0;
    #1; // verify data out valid
    #1; // verify errors; we should get out because we lost bits
    assert(error_led_o == 1'd1); // we should get an error
    #1;
    #1;
    #1;
    #1;
    reset_i = 1'b1;
    #1;
    reset_i = 1'b0;

    ////////////////////////////
    ////   Error on write    ////
    ////////////////////////////
    // in this section we confirm that upon a write error the FSM goes into error state
    assert(i2c_din_o == 8'b0);
    assert(i2c_command_byte_o == 8'b0);
    assert(i2c_slave_addr_o == 8'b0);
    assert(i2c_num_bytes_o == 8'b0);
    #1;
    assert(i2c_din_o == 8'b0); // assert that correct data is going to be written
    assert(i2c_command_byte_o == 8'h6b);
    assert(i2c_slave_addr_o == {7'h68, 1'b0});
    assert(i2c_num_bytes_o == 8'd2);
    #1;
    assert(i2c_write_o == 1'b1); // assert write stobe
    #1;
    i2c_busy_i = 1'b1; // assert that we are waiting for chip to be not busy
    assert(i2c_write_o == 1'b0);
    assert(error_led_o == 1'b0);
    #1;
    i2c_busy_i = 1'b1; // assert that we are waiting for chip to be not busy
    assert(i2c_write_o == 1'b0);
    assert(error_led_o == 1'b0);
    #1;
    i2c_busy_i = 1'b0; // make sure we have not errored out yet because we have an ack and no errors
    assert(error_led_o == 1'b0);
    i2c_write_done_i = 1'b1;
    i2c_arb_lost_i = 1'b1;
    #1; // verify write done
    #1; // verify errors; we should get out because we lost bits
    assert(error_led_o == 1'd1); // we should get an error


    #100 $finish;
  end

endmodule
```