ELC 5396 02 Class Report 3

Elizabeth Kooiman

October 20, 2023

Summary

The purpose of the project was to integrate the accelerometer on the Nexys4 DDR board and turn on one of four LEDs depending on the rotation/orientation of the board. If LED0 is on, then the board is rotated 0 degrees and being held upright similar to a cell phone. LED1 turns on if the board is rotated 90 degrees. LED2 turns on if the board is rotated 180 degrees. LED3 turns on if the board is rotated 270 degrees. . The code for this project can be found in GitHub at https://github.com/elizabethkooiman/Kooiman_SoC.

To start, the SPI module created by Dr. Chu was integrated into the system. To do this, the system verilog files provided for the chu_spi_core and the spi module. Once these files were added to the project, they needed to be integrated into the bus. To do this, the chu_spi_core was instantiated in the mmio module. Slot 9 was reserved for SPI, so this was where the module was placed. The relevent inputs and outputs were connected with reference to the slot. There were four additional external signals that were added to the mmio based on the inclusion of the SPI: spi_sclk, spi_mosi, spi_miso, and spi_ss_n. These signals were added as inputs and outputs to the mmio_sys_vanilla.sv file. Then, these inputs and outputs were added to the mcs_top_vanilla.sv file inside the instantiation of the mmio_sys_vanilla module. Then, the accelerometer signals acl_sclk, acl_mosi, acl_miso, and acl_ss_n signals were put as inputs to the top module and connected to the signals previously listed in the instantiation of the mmio_system_vanilla. Once this was done, the hardware component of the project was complete and the bitsream was generated. The hardware was then exported, including the bitstream.

Then, a platform project was created using the XSA generated from exporting the hardware. Once this was set up and built, an application project was built on top of it. All of the files required for the vanilla system were added to the project. In addition, the spi_core.h and spi_core.cpp files were added. Once this was done, the main sampler provided was used to check the functioning of the desired sensor, in this case the accelerometer. Once it was confirmed the accelerometer was working through the UART messages, a function called gsensor_app() was created to light up LEDs to correspond with the orientation of the board.

Results

The accelerometer gave readings on the x, y, and z axes and these readings were set to correspond with relation to g, 9.8 meters per second squared. Once this was done, an if statement used in the gsensor_app function was used to determine if x or y is close to -g or g which corresponds to the different orientations of the board. The LEDs successfully turned on to indicate the orientation of the board. A video demonstrating the functionality can be found in the GitHub Repository referenced in the summary.

Code

Listing 1: edited mmio module

```
`include "chu_io_map.svh"
module mmio_sys_vanilla
#(
   parameter N_SW = 8,
   parameter N_LED = 8
)
(
   input logic clk,
   input  logic reset,
   // FPro bus
   input  logic mmio_cs,
   input  logic mmio_wr,
   input  logic mmio_rd,
   input  logic [20:0] mmio_addr, // 11 LSB used; 2^6 slots; 2^5 reg each
   input  logic [31:0] mmio_wr_data,
   output logic [31:0] mmio_rd_data,
   // switches and LEDs
   input logic [N_SW-1:0] sw,
   output logic [N_LED-1:0] led,
   // uart
   input logic rx,
   output logic tx,
   //spi
   output logic spi_sclk,
   output logic spi_mosi,
   input  logic spi_miso,
   output logic spi_ss_n  //[2-1:0]
);

   // declaration
//   localparam S = 1;
   logic [63:0] mem_rd_array;
   logic [63:0] mem_wr_array;
   logic [63:0] cs_array;
   logic [4:0] reg_addr_array [63:0];
   logic [31:0] rd_data_array [63:0];
   logic [31:0] wr_data_array [63:0];

//   logic spi_sclk,spi_mosi,spi_miso;
//   logic [S-1:0] spi_ss_n;

   // body
   // instantiate mmio controller
   chu_mmio_controller ctrl_unit
   (.clk(clk),
    .reset(reset),
    .mmio_cs(mmio_cs),
    .mmio_wr(mmio_wr),
    .mmio_rd(mmio_rd),
    .mmio_addr(mmio_addr),
    .mmio_wr_data(mmio_wr_data),
```

```
    .mmio_rd_data(mmio_rd_data),
    // slot interface
    .slot_cs_array(cs_array),
    .slot_mem_rd_array(mem_rd_array),
    .slot_mem_wr_array(mem_wr_array),
    .slot_reg_addr_array(reg_addr_array),
    .slot_rd_data_array(rd_data_array),
    .slot_wr_data_array(wr_data_array)
    );

  // slot 0: system timer
  chu_timer timer_slot0
  (.clk(clk),
    .reset(reset),
    .cs(cs_array[`S0_SYS_TIMER]),
    .read(mem_rd_array[`S0_SYS_TIMER]),
    .write(mem_wr_array[`S0_SYS_TIMER]),
    .addr(reg_addr_array[`S0_SYS_TIMER]),
    .rd_data(rd_data_array[`S0_SYS_TIMER]),
    .wr_data(wr_data_array[`S0_SYS_TIMER])
    );

  // slot 1: UART
  chu_uart uart_slot1
  (.clk(clk),
    .reset(reset),
    .cs(cs_array[`S1_UART1]),
    .read(mem_rd_array[`S1_UART1]),
    .write(mem_wr_array[`S1_UART1]),
    .addr(reg_addr_array[`S1_UART1]),
    .rd_data(rd_data_array[`S1_UART1]),
    .wr_data(wr_data_array[`S1_UART1]),
    .tx(tx),
    .rx(rx)
    );
  //assign rd_data_array[1] = 32'h00000000;

  // slot 2: gpo
  chu_gpo #(.W(N_LED)) gpo_slot2
  (.clk(clk),
    .reset(reset),
    .cs(cs_array[`S2_LED]),
    .read(mem_rd_array[`S2_LED]),
    .write(mem_wr_array[`S2_LED]),
    .addr(reg_addr_array[`S2_LED]),
    .rd_data(rd_data_array[`S2_LED]),
    .wr_data(wr_data_array[`S2_LED]),
    .dout(led)
    );

  // slot 3: gpi
  chu_gpi #(.W(N_SW)) gpi_slot3
  (.clk(clk),
    .reset(reset),
```

```
    .cs(cs_array[`S3_SW]),
    .read(mem_rd_array[`S3_SW]),
    .write(mem_wr_array[`S3_SW]),
    .addr(reg_addr_array[`S3_SW]),
    .rd_data(rd_data_array[`S3_SW]),
    .wr_data(wr_data_array[`S3_SW]),
    .din(sw)
    );

    //slot 9: spi
    chu_spi_core #(.S(1)) spi_slot9  // width (# bits) of output port
   (
    .clk(clk),
    .reset(reset),
    // slot interface
    .cs(cs_array[`S9_SPI]),
    .read(mem_rd_array[`S9_SPI]),
    .write(mem_wr_array[`S9_SPI]),
    .addr(reg_addr_array[`S9_SPI]),
    .wr_data(wr_data_array[`S9_SPI]),
    .rd_data(rd_data_array[`S9_SPI]),
    // external signal
    .spi_sclk(spi_sclk),
    .spi_mosi(spi_mosi),
    .spi_miso(spi_miso),
    .spi_ss_n(spi_ss_n)
   );

   // assign 0's to all unused slot rd_data signals
   generate
      genvar i;
      for (i=10; i<64; i=i+1) begin:  unused_slot_gen
         assign rd_data_array[i] = 32'hffffffff;
      end
      for (i=4; i<9; i=i+1) begin:  unused_slot_gen2
         assign rd_data_array[i] = 32'hffffffff;
      end
   endgenerate
endmodule


   // slot interface
   //output wire [63:0] slot_cs_array,
   //output wire [63:0] slot_mem_rd_array,
   //output wire [63:0] slot_mem_wr_array,
   //output wire [4:0]  slot_reg_addr_array [0:63],
   //input wire  [31:0] slot_rd_data_array [63:0],
   //output wire [31:0] slot_wr_data_array [63:0]
   // verilog not allow 2d-array port
   //output wire [64*4-1:0]  slot_reg_addr_1d,
   //input wire  [64*32-1:0] slot_rd_data_1d,
   //output wire [64*32-1:0] slot_wr_data_1d

//entity mmio_sys_vanilla is
```

```
//   generic(
//     N_LED: integer;
//     N_SW: integer
// );
//   port(
//      -- FPro bus
//      clk          : in  std_logic;
//      reset        : in  std_logic;
//      mmio_cs      : in  std_logic;
//      mmio_wr      : in  std_logic;
//      mmio_rd      : in  std_logic;
//      mmio_addr    : in  std_logic_vector(20 downto 0); -- only 11 LSBs
   used
//      mmio_wr_data : in  std_logic_vector(31 downto 0);
//      mmio_rd_data : out std_logic_vector(31 downto 0);
//      -- switches and LEDs
//      sw           : in  std_logic_vector(N_SW-1 downto 0);
//      led          : out std_logic_vector(N_LED-1 downto 0);
//      -- uart
//      rx           : in  std_logic;
//      tx           : out std_logic
//   );
//end mmio_sys_vanilla;
//
//architecture arch of mmio_sys_vanilla is
//   signal cs_array       : std_logic_vector(63 downto 0);
//   signal reg_addr_array : slot_2d_reg_type;
//   signal mem_rd_array   : std_logic_vector(63 downto 0);
//   signal mem_wr_array   : std_logic_vector(63 downto 0);
//   signal rd_data_array  : slot_2d_data_type;
//   signal wr_data_array  : slot_2d_data_type;
//
//begin
//   --*****************************************************************
//   --  MMIO controller instantiation
//   --*****************************************************************
//   ctrl_unit : entity work.chu_mmio_controller
//      port map(
//         -- FPro bus interface
//         mmio_cs            => mmio_cs,
//         mmio_wr            => mmio_wr,
//         mmio_rd            => mmio_rd,
//         mmio_addr          => mmio_addr,
//         mmio_wr_data       => mmio_wr_data,
//         mmio_rd_data       => mmio_rd_data,
//         -- 64 slot interface
//         slot_cs_array      => cs_array,
//         slot_reg_addr_array => reg_addr_array,
//         slot_mem_rd_array  => mem_rd_array,
//         slot_mem_wr_array  => mem_wr_array,
//         slot_rd_data_array => rd_data_array,
//         slot_wr_data_array => wr_data_array
//      );
//
```

```
//    --************************************************************
//    -- IO slots instantiations
//    --************************************************************
//    -- slot 0: system timer
//    timer_slot0 : entity work.chu_timer
//       port map(
//          clk           => clk,
//          reset         => reset,
//          cs            => cs_array[S0_SYS_TIMER],
//          read          => mem_rd_array[S0_SYS_TIMER],
//          write         => mem_wr_array[S0_SYS_TIMER],
//          addr          => reg_addr_array[S0_SYS_TIMER],
//          rd_data       => rd_data_array[S0_SYS_TIMER],
//          wr_data       => wr_data_array[S0_SYS_TIMER]
//       );
//
//    -- slot 1: uart1
//    uart1_slot1 : entity work.chu_uart
//       generic map(FIFO_DEPTH_BIT => 6)
//       port map(
//          clk     => clk,
//          reset   => reset,
//          cs      => cs_array[S1_UART],
//          read    => mem_rd_array[S1_UART],
//          write   => mem_wr_array[S1_UART],
//          addr    => reg_addr_array[S1_UART],
//          rd_data => rd_data_array[S1_UART],
//          wr_data => wr_data_array[S1_UART],
//          -- external signals
//          tx      => tx,
//          rx      => rx
//       );
//
//    -- slot 2: GPO for LEDs
//    gpo_slot2 : entity work.chu_gpo
//       generic map(W => N_LED)
//       port map(
//          clk     => clk,
//          reset   => reset,
//          cs      => cs_array[S2_LED],
//          read    => mem_rd_array[S2_LED],
//          write   => mem_wr_array[S2_LED],
//          addr    => reg_addr_array[S2_LED],
//          rd_data => rd_data_array[S2_LED],
//          wr_data => wr_data_array[S2_LED],
//          -- external signal
//          dout    => led
//       );
//
//    -- slot 3: input port for switches
//    gpi_slot3 : entity work.chu_gpi
//       generic map(W => N_SW)
//       port map(
//          clk     => clk,
```

```
//          reset   => reset,
//          cs      => cs_array[S3_SW],
//          read    => mem_rd_array[S3_SW],
//          write   => mem_wr_array[S3_SW],
//          addr    => reg_addr_array[S3_SW],
//          rd_data => rd_data_array[S3_SW],
//          wr_data => wr_data_array[S3_SW],
//          -- external signal
//          din     => sw
//       );
//
//    -- assign 0's to all unused slot rd_data signals
//    gen_unused_slot : for i in 4 to 63 generate
//       rd_data_array(i) <= (others => '0');
//    end generate gen_unused_slot;
//end arch;
```

Listing 2: edited top module

```
module mcs_top_vanilla
#(parameter BRG_BASE = 32'hc000_0000)
(
   input logic clk,
   input logic reset_n,
   // switches and LEDs
   input logic [15:0] sw,
   output logic [15:0] led,
   // uart
   input logic rx,
   output logic tx,
   //spi
   output logic acl_sclk,
   output logic acl_mosi,
   input  logic acl_miso,
   output logic acl_ss_n   //[2-1:0]
);

   // declaration
   logic clk_100M;
   logic reset_sys;
   // MCS IO bus
   logic io_addr_strobe;
   logic io_read_strobe;
   logic io_write_strobe;
   logic [3:0] io_byte_enable;
   logic [31:0] io_address;
   logic [31:0] io_write_data;
   logic [31:0] io_read_data;
   logic io_ready;
   // fpro bus
   logic fp_mmio_cs;
   logic fp_wr;
   logic fp_rd;
   logic [20:0] fp_addr;
```

```
  logic [31:0] fp_wr_data;
  logic [31:0] fp_rd_data;

  // body
  assign clk_100M = clk;                    // 100 MHz external clock
  assign reset_sys = !reset_n;

  //instantiate uBlaze MCS
  cpu cpu_unit (
   .Clk(clk_100M),                         // input wire Clk
   .Reset(reset_sys),                      // input wire Reset
   .IO_addr_strobe(io_addr_strobe),        // output wire IO_addr_strobe
   .IO_address(io_address),                // output wire [31 : 0] IO_address
   .IO_byte_enable(io_byte_enable),        // output wire [3 : 0]
      IO_byte_enable
   .IO_read_data(io_read_data),            // input wire [31 : 0]
      IO_read_data
   .IO_read_strobe(io_read_strobe),        // output wire IO_read_strobe
   .IO_ready(io_ready),                    // input wire IO_ready
   .IO_write_data(io_write_data),          // output wire [31 : 0]
      IO_write_data
   .IO_write_strobe(io_write_strobe)       // output wire IO_write_strobe
  );

  // instantiate bridge
  chu_mcs_bridge #(.BRG_BASE(BRG_BASE)) bridge_unit (.*, .fp_video_cs());

  // instantiated i/o subsystem
  mmio_sys_vanilla #(.N_SW(16),.N_LED(16)) mmio_unit (
  .clk(clk),
  .reset(reset_sys),
  .mmio_cs(fp_mmio_cs),
  .mmio_wr(fp_wr),
  .mmio_rd(fp_rd),
  .mmio_addr(fp_addr),
  .mmio_wr_data(fp_wr_data),
  .mmio_rd_data(fp_rd_data),
  .sw(sw),
  .led(led),
  .rx(rx),
  .tx(tx),
  .spi_sclk(acl_sclk),
  .spi_mosi(acl_mosi),
  .spi_miso(acl_miso),
  .spi_ss_n(acl_ss_n)
 );
endmodule
```

Listing 3: main file and application functions

```
/*****************************************************************//**
 * @file main_sampler_test.cpp
 *
 * @brief Basic test of nexys4 ddr mmio cores
```

```
 *
 * @author p chu
 * @version v1.0: initial release
 ********************************************************************/

// #define _DEBUG
#include "chu_init.h"
#include "gpio_cores.h"
#include "spi_core.h"

/**
 * blink once per second for 5 times.
 * provide a sanity check for timer (based on SYS_CLK_FREQ)
 * @param led_p pointer to led instance
 */
void timer_check(GpoCore *led_p) {
   int i;

   for (i = 0; i < 5; i++) {
      led_p->write(0xffff);
      sleep_ms(500);
      led_p->write(0x0000);
      sleep_ms(500);
      debug("timer check - (loop #)/now: ", i, now_ms());
   }
}

/**
 * check individual led
 * @param led_p pointer to led instance
 * @param n number of led
 */
void led_check(GpoCore *led_p, int n) {
   int i;

   for (i = 0; i < n; i++) {
      led_p->write(1, i);
      sleep_ms(100);
      led_p->write(0, i);
      sleep_ms(100);
   }
}

/**
 * leds flash according to switch positions.
 * @param led_p pointer to led instance
 * @param sw_p pointer to switch instance
 */
```

```
void sw_check(GpoCore *led_p, GpiCore *sw_p) {
   int i, s;

   s = sw_p->read();
   for (i = 0; i < 30; i++) {
      led_p->write(s);
      sleep_ms(50);
      led_p->write(0);
      sleep_ms(50);
   }
}

/**
 * uart transmits test line.
 * @note uart instance is declared as global variable in chu_io_basic.h
 */
void uart_check() {
   static int loop = 0;

   uart.disp("uart test #");
   uart.disp(loop);
   uart.disp("\n\r");
   loop++;
}


/**
 * Test adxl362 accelerometer using SPI
 */

void gsensor_check(SpiCore *spi_p, GpoCore *led_p) {
   const uint8_t RD_CMD = 0x0b;
   const uint8_t PART_ID_REG = 0x02;
   const uint8_t DATA_REG = 0x08;
   const float raw_max = 127.0 / 2.0;  //128 max 8-bit reading for +/-2g

   int8_t xraw, yraw, zraw;
   float x, y, z;
   int id;

   spi_p->set_freq(400000);
   spi_p->set_mode(0, 0);
   // check part id
   spi_p->assert_ss(0);     // activate
   spi_p->transfer(RD_CMD);  // for read operation
   spi_p->transfer(PART_ID_REG);  // part id address
   id = (int) spi_p->transfer(0x00);
   spi_p->deassert_ss(0);
```

```
    uart.disp("read ADXL362 id (should be 0xf2): ");
    uart.disp(id, 16);
    uart.disp("\n\r");
    // read 8-bit x/y/z g values once
    spi_p->assert_ss(0);     // activate
    spi_p->transfer(RD_CMD);  // for read operation
    spi_p->transfer(DATA_REG);  //
    xraw = spi_p->transfer(0x00);
    yraw = spi_p->transfer(0x00);
    zraw = spi_p->transfer(0x00);
    spi_p->deassert_ss(0);
    x = (float) xraw / raw_max;
    y = (float) yraw / raw_max;
    z = (float) zraw / raw_max;
    uart.disp("x/y/z axis g values: ");
    uart.disp(x, 3);
    uart.disp(" / ");
    uart.disp(y, 3);
    uart.disp(" / ");
    uart.disp(z, 3);
    uart.disp("\n\r");
}
 void gsensor_app(SpiCore *spi_p, GpoCore *led_p) {
    const uint8_t RD_CMD = 0x0b;
    const uint8_t PART_ID_REG = 0x02;
    const uint8_t DATA_REG = 0x08;
    const float raw_max = 127.0 / 2.0;  //128 max 8-bit reading for +/-2g

    int8_t xraw, yraw, zraw;
    float x, y, z;
    int id;

    spi_p->set_freq(400000);
    spi_p->set_mode(0, 0);
    // check part id
    spi_p->assert_ss(0);     // activate
    spi_p->transfer(RD_CMD);  // for read operation
    spi_p->transfer(PART_ID_REG);  // part id address
    id = (int) spi_p->transfer(0x00);
    spi_p->deassert_ss(0);
    uart.disp("read ADXL362 id (should be 0xf2): ");
    uart.disp(id, 16);
    uart.disp("\n\r");
    // read 8-bit x/y/z g values once
    spi_p->assert_ss(0);     // activate
    spi_p->transfer(RD_CMD);  // for read operation
    spi_p->transfer(DATA_REG);  //
    xraw = spi_p->transfer(0x00);
```

```
    yraw = spi_p->transfer(0x00);
    zraw = spi_p->transfer(0x00);
    spi_p->deassert_ss(0);
    x = (float) xraw / raw_max;
    y = (float) yraw / raw_max;
    z = (float) zraw / raw_max;
    uart.disp("x/y/z axis g values: ");
    uart.disp(x, 3);
    uart.disp(" / ");
    uart.disp(y, 3);
    uart.disp(" / ");
    uart.disp(z, 3);
    uart.disp("\n\r");
    if(y <= -0.8){
        //board is flat
        led_p->write(0x0001);
    }
    else if(x <=-0.8){
        led_p->write(0x0002);
    }
    else if ((y <= 1.2)&&(y > 0.8)){
        led_p->write(0x0004);
    }
    else if ((x <= 1.2)&&(x > 0.8)){
                led_p->write(0x0008);
        }
    else{
        led_p ->write(0x0000);
    }
 }

/**
 * core test
 * @param led_p pointer to led instance
 * @param sw_p pointer to switch instance
 */
void show_test_id(int n, GpoCore *led_p) {
   int i, ptn;

   ptn = n; //1 << n;
   for (i = 0; i < 20; i++) {
      led_p->write(ptn);
      sleep_ms(30);
      led_p->write(0);
      sleep_ms(30);
   }
}
```

```
GpoCore led(get_slot_addr(BRIDGE_BASE, S2_LED));
GpiCore sw(get_slot_addr(BRIDGE_BASE, S3_SW));
SpiCore spi(get_slot_addr(BRIDGE_BASE, S9_SPI));


int main() {
   //uint8_t id, ;

   timer_check(&led);
   while (1) {
           gsensor_app(&spi, &led);
   } //while
} //main
```