Lab3: Auto-Adjust Brightness Controller

Objective: Design an auto-adjust brightness core. Students will develop a Serial Peripheral Interface (SPI) controller to interface the FPGA (master device) to an ambient light *sensor chip*. The Zedboard's LEDs will auto adjust their illumination based on the lighting conditions applied to the *sensor chip*. Students will use Bus Functional Models (BFM) and testbenches to verify their design thoroughly, and on the day of their demo, test that their design is correct using the *physical* sensor chip.

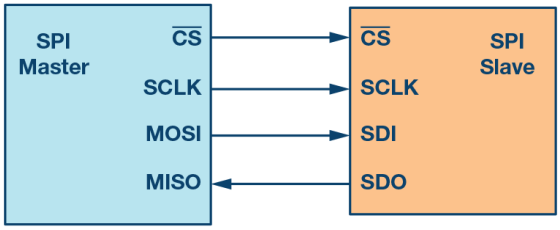

Fig. 1 (a): Structural Overview of a SPI Controller (Wikipedia)

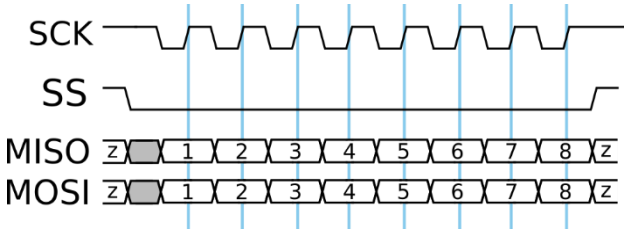

Fig. 1 (b): Timing Diagram of the SPI protocol (Wikipedia)

1. Serial Peripheral Interface (SPI) Background Information

Serial Peripheral Interface (SPI) is a popular synchronous, serial communication protocol for interfacing chips. SPI is a defacto standard, meaning there is no official documentation describing the exact protocol. Accordingly, many variations of SPI controllers exist depending on a chip's interfacing requirements. Regardless of the variations, there are 3 to 4 standard pins used by the SPI communication protocol as suggested in Fig. 1 (a):

- **Chip Select (CS)** aka Slave Select (SS). Master chip uses this line to select an available slave.
  - CS is commonly defaulted to active-low logic (see Fig 1. (b)). When CS is high, the **slave chip is idle** and a master may take control of it. When CS is low, the slave chip is busy and serving another transaction.
- **Serial Clock (SCLK)**. SCLK is used as a clock source to transmit serial data synchronously. The master core generates the SCLK once the slave is available and *CS* is pulled low.
  - SCLK is high when idle. When CS is pulled low by the master to transmit data to the selected save, the Master generates a clock pulse for SCLK for a predefined number of clock cycles (based on slave requirements and frequency). Once the data has been transmitted, SCLK is pulled high to indicate that the master is back in the idle state.
- **MISO** – **M**aster **I**n, **S**lave **O**ut. 1bit data wire used to communicate serially from slave to master.
  - When the CS is pulled low, the slave will take control of the MISO data line, transmitting data serially to the master, 1 bit per clock cycle (MSB to LSB) using SCLK. Once the data has been transmitted, the MISO line transitions into a *floating state*.
    - Data is written on the falling clock edge of SCLK
    - Data is read on the rising clock edge of SCLK
- **MOSI** – **M**aster **O**ut, **S**lave **I**n. Data line that is used to send data from the master to a slave. Identical properties to MISO however transmitting data in the opposite direction, from master to slave.

Since there is only one SCLK, if data must be sent and received concurrently by the master and slave, the MISO and MOSI are both used, guided by SCLK. Please refer to your tutorial slides for more details regarding SPI and this lab assignment. Below are general comments and design strategies to help you complete this lab.

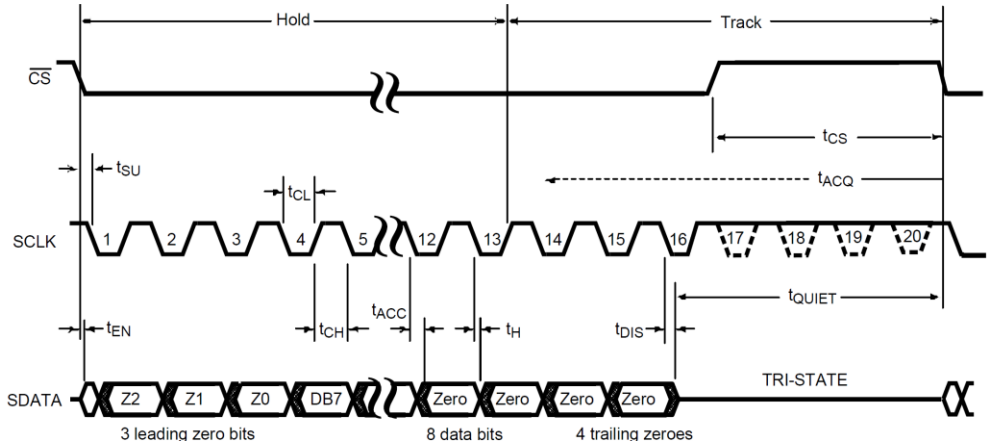

Fig. 2: Timing Diagram of the Ambient Light Sensor (pp. 7 of Adc081s021 datasheet)

1.1 Light Sensor: Master-only SPI Controller Design

The Adc081s021.pdf datasheet provides specifications of the Ambient Light Sensor. The chip contains an on-board light sensor and Analog-to-Digital Converter (ADC) that provides digital output. Pp. 7 provides the timing characteristics necessary for our SPI controller (master designed on FPGA) to control the Ambient Light Sensor (external slave chip). Pay close attention to the details provided in Fig. 2:

- CS is pulled low by the master to transmit. SCLK will start pulsing at the required frequency.
- Data is written by the sensor on the falling edge of SCLK and read by the master on the rising edge. 16bits in total will be sent, however only 8 bits of the data transmitted represent the actual sensor data (DB7..DB0).
- 3 SCLK cycle delay (Z2, Z1, Z0 i.e. bits 1 - 3) will appear on the MISO line after CS is pulled low. Thereafter the 8bit value of interest from the sensor will appear on the MISO (SDATA) line, written MSB to LSB. Thereafter 4 trailing zeros (Bits 12 – 16) will appear on SDATA until the CS line is pulled high by the master. SDATA will thereafter be tri-stated by the sensor chip. For this lab, *we will only require MISO communication*.

1.2 Controller design

1.2.1 Prescaler, constants and counters: Like previous labs, you will require a prescaler to generate the SCLK required of the slave/sensor chip. Create a counter circuit that appropriately scales the FPGA clock (100MHz) to the sensor's clock running at 4MHz. Since you will likely be coding the clock as sclk <= not sclk (inverting the clock every X cycles), consider the clock cycles to wait for **half of a** SCLK period.

If designed in a proper way, your prescaler from Lab1 may be used in this lab. Set the sensor clock and fpga clock properly by setting the generic values. Instantiate the component into your design.

Do not hard code any values in your VHDL design. Ensure that your design is scalable and can easily be applied to any FPGA by setting your ENTITY's generic values. Below is the suggested ENTITY declaration to be used for your spi_controller.vhd.

```
entity spi_master is
  generic (
    clk_hz : integer; --FPGA clock
    total_bits : integer –total bits tx by sensor chip
    sclk_hz : integer); --sensor's frequency
  port (
    --fpga system
    clk : in std_logic;
    rst : in std_logic;

    -- slave chip
    cs : out std_logic;
    sclk : out std_logic;
    miso : in std_logic;

    -- Internal interface when obtaining data back from slave chip
    ready : in std_logic;
    valid : out std_logic;
    data : out std_logic_vector(7 downto 0));
end spi_master;
```

1.2.2. FSM design:

Create an FSM consisting of two states: IDLE and TRANSMISSION.

IDLE state: awaits for the input ready signal (see above). When asserted, the FSM transitions to TRANSMISSION to start the SPI protocol and de-assert CS.

TRANSMISSION state: 1) generates sclk (use counters to generate pulse appropriately and only when required), 2) handles the 16bits transmitted by the sensor chip, obtain correct 8bit data, use a right shift register to store sensor data. Once completed, 3) pulls sclk and cs high and transitions back to the IDLE state.

Ensure your FSM behaves identically to Fig.1 and as expected using the template testbench and Bus Functional Model (BFM) provided. Refer to the tutorial for specifics on the timing behaviour and the BFM and tb files for verification. Pay special attention to the behaviour of the sclk and CS lines.

1.2.3 Sampling the MISO data:

The SDATA (miso input) input to your controller is metastable. Use a synchronizer to stabilize the data and avoid issues of metastability. Note that SCLK does NOT need to be stabilized since your controller (i.e. the master) is generating the clock for the slave to the send data.

As stated above, use an 8bit shift register to acquire the 1bit data sent from the sensor on SDATA (miso). Recall that the only valid data is sent on cycles 3 – 11, however your synchronizer may contribute to delay during sampling. You must account for this delay in your design. Use the BFM and testbenches provided to you to appropriately ensure data is being received correctly (more details provided below).

Once the 8bit SDATA packet is sampled appropriately after "cycle 11", data may be output to the "data" port on the next cycle with a **pulsed valid bit**.

1.2.4 Design Verification: Your SPI controller master will be deployed on the FPGA and interfaced with a real Ambient Light sensor chip during your lab demo. To assist you through the development of your SPI controller to ensure that your design works correctly come demo day, a Bus Functional Model (BFM) of the sensor has been provided to you (als_bfm.vhd), in addition to a self-checking testbench template (spi_controller_tb.vhd). Use this to thoroughly verify your SPI controller and ensure it will behave as intended when integrated into your overall system. Run the testbench every time you complete a portion of your design to ensure that the VHDL is behaving appropriately as expected. Catch your design bugs as they happen.

2. Lab Assignment: Auto-Adjust Brightness Controller

The objective of the lab assignment is to design an Auto-Adjust Brightness Controller (AABC). The controller will adjust the brightness of the onboard LEDs based on the ambient lighting conditions sent from the external sensor via SPI. The brighter the light that shines on the sensor, the brighter the LEDs on the Zedboard will shine. Conversely the darker the ambient lighting conditions, the dimmer the LEDs.

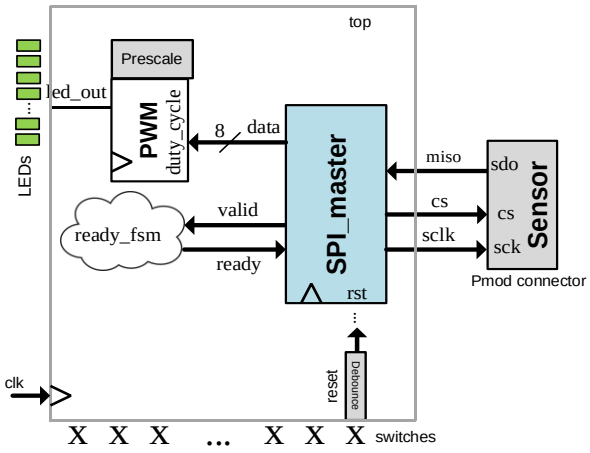

Fig. 3: Block Diagram of the Auto-Adjust Brightness Controller (AABC)

2.1 Component Integration

To achieve the desired behaviour specified for the AABC, you must integrate your 1) spi_controller from section 1.2 and 2) pwm components designed in lab1. Use the data obtained from the sensor, i.e. the 8b packet assembled by the spi_master, and apply this data as your duty cycle input to the pwm controller. The output of the PWM controller will be used to light the Zedboard's LEDs according to the sensor data. To ensure that the LEDs are switched on and off for proper visibility, ensure that another prescaler is instantiated to light the LEDs appropriately.

The top.vhd template, i.e. top-level entity VHDL template, has been provided to you. Instantiate all your components inside this file as suggested in Fig. 3. Note that some details are omitted on purpose in in the figure and left for you to implement as the hardware designer. Also included in the top-level entity is a process entitled ready_fsm. This provides simple counter logic which periodically asserts the ready signal for your spi_controller to initiate communication with the light sensor.

A debouncer circuit has also been provided to you. Open the file and understand the behaviour of the VHDL. Use it to "debounce" your reset switch signal from mechanical jitter. Apply an appropriate delay using the generic parameter provided (time_out).

2.2. Design Verification

Create your own testbench to verify that your AABC behaves as intended. Integrate the BFM into your testbench to mimic the sensor behaviour, and thoroughly verify your circuit. Ensure that as your BFM provides values closer to 255, the duty cycle input and output by the PWM controller is also reflective of these values.

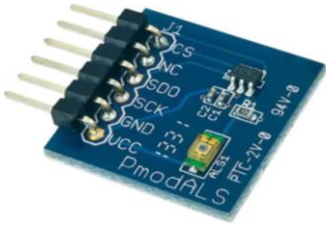

Fig. 4: (a) Ambient Light Sensor with PMOD connector

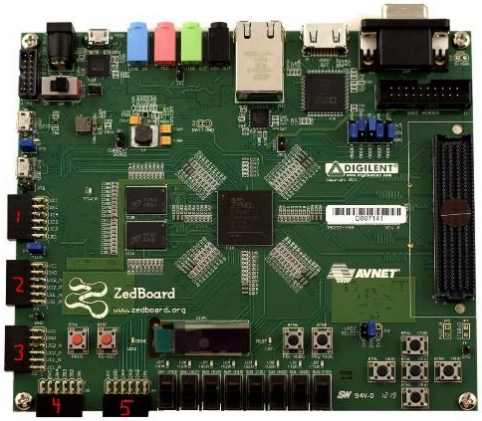

Fig. 4: b) Labelled PMOD connectors on Zedboard

2.3 Design Constraint Files & Pin Assignments

The ambient light sensor chip is shown in Fig. 4a. This chip will fit nicely into one of the five pmod connectors on the zedboard as numbered in red in Fig. 4b.

Create your pin constraint file, Xilinx Design Constraint (xdc) file, called top.xdc. Select one of the pmod connectors and appropriately assign the pins into your top.xdc. As seen in Fig. 4(a), the VCC and GND pins on the sensor chip must properly align to the VCC and GND pin on the Zedboard's pmod connector.

Use the UG18 data sheet and the xdc files from your previous lab as templates to guide you in appropriately assigning the pins. Assign the following pins in your top-level design:

- clk
- rst (assign to one of the switches)
- cs
- sclk
- miso
- led_out

Presentation/Deliverables/Group Work **Due Friday, March 18th @ 11:59pm**

Demo: Please sign up for your demo using Canvas' Calendar. Both members MUST be present for the demo, else a mark of zero will be assigned to the demo portion of this project.

You and your partner must demo your top-level design to the TA during the scheduled time and answer a series of questions. The demo process should take 10 minutes max. *If something in your design does not work properly, clearly state the blunder else you will be heavily penalized for the omission.*

Deliverables uploaded to Canvas as one Zip:

- **For those working in partners**: Include an **individual report**. Clearly outline and describe each of the contributions you have made to this project. Marks will be assigned accordingly to workload distribution. You and your partner will not be assigned the same marks for uneven work distribution.

- Your top-level Xilinx project as a folder "Xilinx". Include the Xilinx bitstream file, xdc etc for the hardware design

- "Code" folder consisting of your a) "SPI_controller" and b) "top" subfolder. Each of these subfolders contain their respective VHDL files, testbenches, DO scripts etc required for automatically verifying Part1 (spi controller design) and Part2 (integration of SPI controller and pwm components) of this lab independently. Ensure that you include your "work" folder for each.
    - Consider the TA will be using VS Code and automatically running your testbenches using Modelsim. Ensure you have the proper DO scripts in each folder to simulate the top-level files for each part.
    - Keep your code clean and concise, with proper formatting. Marks will be assigned accordingly. Use comments so the TAs can follow your work. Any vague code will be marked as interpreted by your TA.

- A **"debug" folder** that contains screenshots of the waveforms used to verify your component designs. Label the screenshots. Ensure subfolders included are organized and well-labelled.

- A README file commenting on your submission organization, folders, and files so that the TA may navigate and find everything required of this lab and its respective parts.