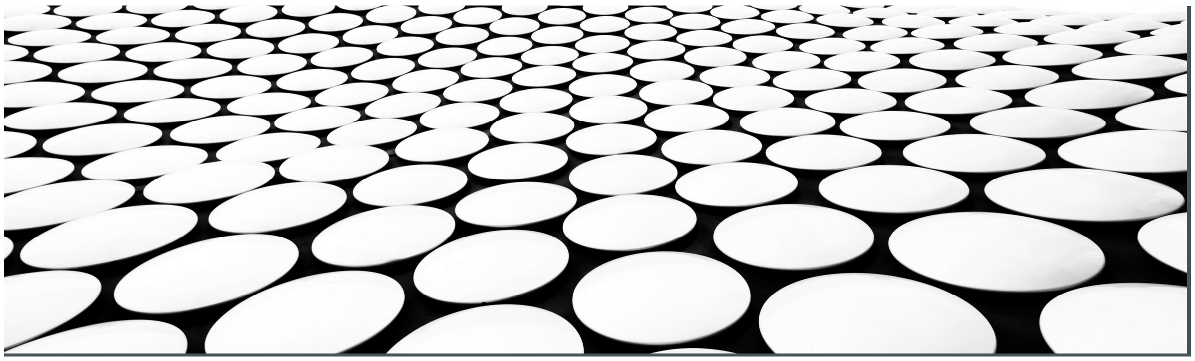
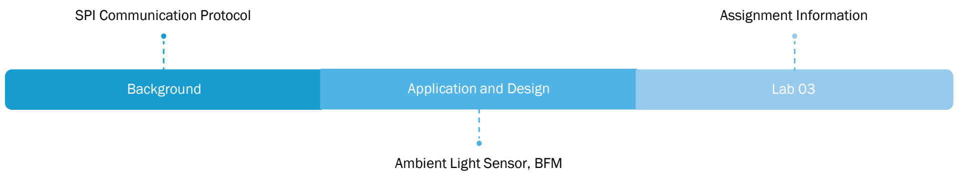
TUTORIAL #4: AUTO-ADJUST BRIGHTNESS CONTROLLER DESIGN

ENSC462: ADVANCED DIGITAL SYSTEMS – APPLICATION OF SERIAL PERIPHERAL INTERFACE (SPI) PROTOCOL



1

LECTURE OVERVIEW



2

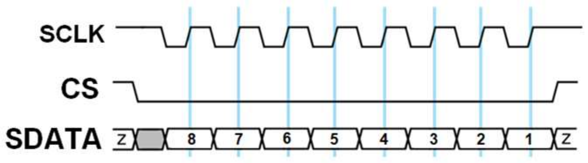
SERIAL PERIPHERAL INTERFACE (SPI)

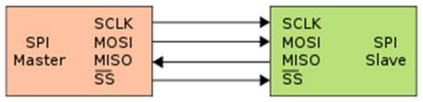
BRIEF THEORY

3

SPI COMMUNICATION PROTOCOL

- Simple synchronous, serial communication protocol for interfacing Integrated Circuits (ICs)
  - Ex: Memory, sensors, LCDs, OLEDs, SD cards, ADCs
  - USB is more popular, however more complicated to interface

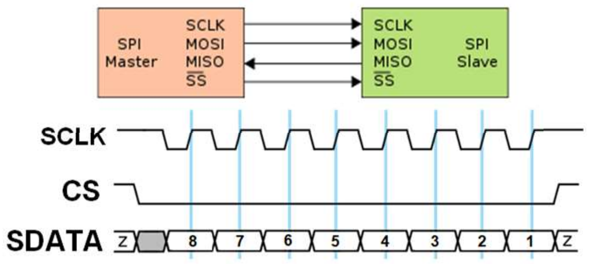




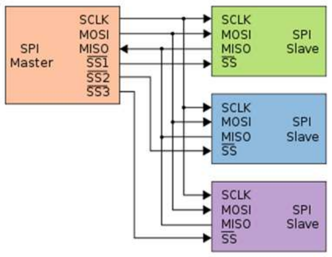
- De-facto standard
  - No official document describing the protocol exactly. Therefore there are many variations depending on the interfacing requirements
- 4 main pins used:
  - 1) **SCLK** - Serial Clock. '1' inactive, pulsed = active clock
  - 2) **CS** – Chip Selected. '1' inactive, '0' chip is serving a transmission (active-low). Aka *Slave Select (SS)*
  - **SDATA** - serial data being sent/received wrt SCLK. Depending on what chip is sending and receiving data, SDATA is called:
    - 3) Master In, Slave Out (MISO)
    - 4) Master Out, Slave In (MOSI)
      - may have both MISO and MOSI, called "full-duplex"
  - Master is typically an FPGA, microcontroller, CPU etc
  - Slave is the IC, accelerator etc
  - Data is typically sent MSB to LSB

4

SPI COMMUNICATION PROTOCOL



- Master initiates the transaction to select slave if available. Master pulls CS low (0) + generates SCLK pulse according to frequency required by slave
- Slave uses SCLK to write on SDATA line (MISO) and or read from master using SDATA (MOSI)
  - Write on falling edge, read on rising edge
- Multiple variations of exact specifications of this protocol

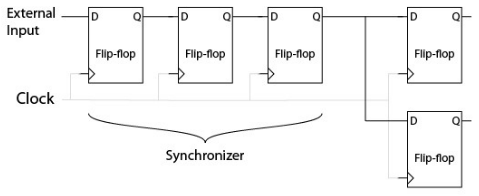


- We will assume a design CPOL = 1 and CPHA=1
- SPI bus has single master device with one or more slaves connected
  - Mutually exclusive, master communicates with 1 slave at a time
- Clock is generated/output by master chip
- MISO/MOSI is 'Z' (tri-stated) when the device is not selected or transmitting serial data

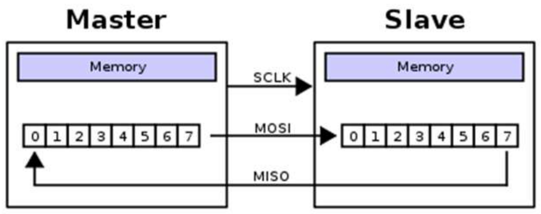
5

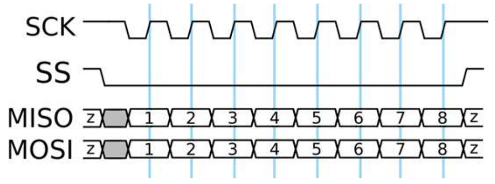
SPI CONTROLLER DESIGN

- Use a shift register to accept the serial data in
  - SCLK rising edge
  - Right shift, append incoming bit to the MSB
- Avoid metastability
- SCLK does **not** need to be stabilized –generated by master, reference clock (FPGA).
- MISO (slave) data may be metastable. Slave running at a different frequency than Master, attempting to tx/rx on another clock frequency. Use synchronizer circuit
  - Consider this when designing a SPI controller



- How does the slave read MOSI values?





6

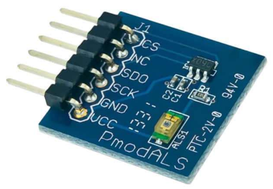
SPI MASTER + AMBIENT LIGHT SENSOR

APPLICATION AND DESIGN

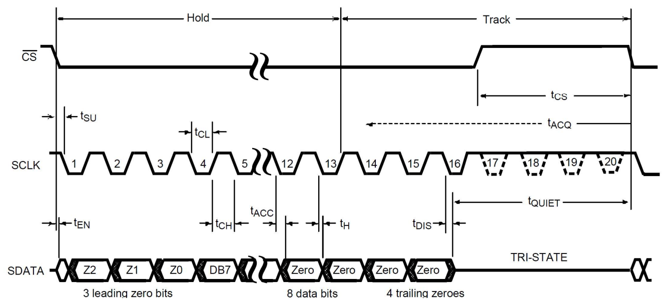
7

DIGILENT AMBIENT LIGHT SENSOR

- Senses amount of light shone onto the chip
- Contains on-board Texas Instrument ADC (8b resolution)
  - Outputs 8b digital values indicating the amount of light exposure to the sensor.
  - 0 = low light exposure , 255 = high light exposure



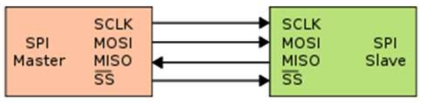
- Timing diagram pp. 7 of datasheet

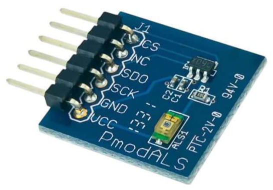


- **16b in total transmitted** on MOSI
- First 3b = 0 + 8 bits data (MSB to LSB) + 4 trailing 0 bits
- Tri-stated at all other times
- Timing considerations modelled in BFM

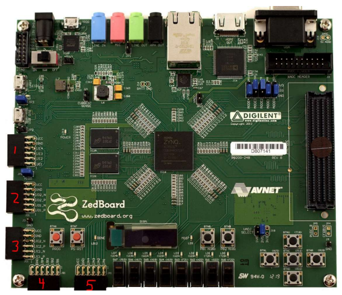
8

DIGILENT AMBIENT LIGHT SENSOR





- Master = FPGA
- Slave = ambient light sensor. Provides data based on light sensor illumination
- MISO connection only, no MOSI
- Create a SPI master controller – reads 8b values coming from sensor when master requests information from sensor (Chip Select (SS in diagram above)



Pinout:

- **CS – chip select**
- **NC – no connection**
- **SDO – MISO data line**
- **SCK – serial clock SCLK**
- **GND**
- **VCC**
- Pmod connector on Zedboard

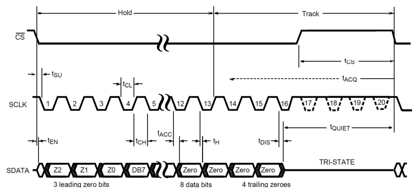
9

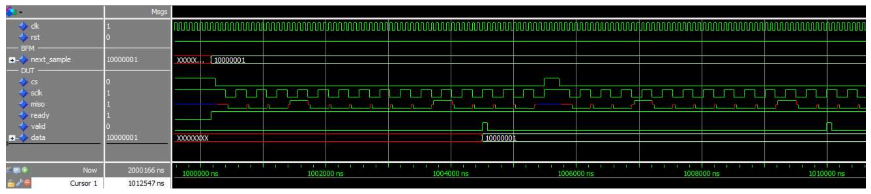
BUS FUNCTIONAL MODEL (BFM)

- You will need to verify your hardware controller without the sensor during design and development. How do you accomplish this?
  - Bus Functional Model (BFM)
- BFM is a behavioural model that "mimics" your DUT's interface circuitry
- BFM's emulate transactions that would occur on bus lines
  - Generates test patterns and control signals as your DUT would typically be exposed to
- Used to successfully verify and simulate your DUT without need to physically connect the hardware every time to the bus/sensor etc

Must be compiled as VHDL-2008

- Integrate the BFM in your testbench
- Interface it with your DUT in the testbench
- Verify your design
- 100% models the Ambient Light Sensor, get you ready for demo day





Signal	Msgs
clk	1
rst	0
next_sample	10000001
cs	0
sclk	1
miso	1
ready	1
valid	0
data	10000001
Now	2000166 ns
Cursor 1	1012547 ns

```
XXXXX... 10000001
XXXXXXXX 10000001
1000000 ns  1002000 ns  1004000 ns  1006000 ns  1008000 ns  1010000 ns
```

10

SELF-CHECKING TESTBENCH

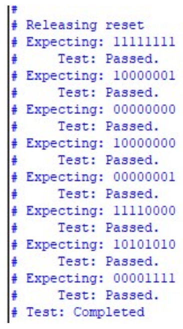
- Initial design phase: Use a preliminary waveform to verify design.
- Thereafter: Use a Self-Checking Testbench (SCT)
- SCT: VHDL testbench that automatically verifies the correctness of your DUT without relying on manually inspecting the waveform
- Use messages, printed to the terminal, such as “Pass” or “Fail” and the values expected/obtained to pinpoint and inspect issues.
- Provided with VHDL procedures “print” and “verify”
- Must be compiled as VHDL-2008

```
procedure verify(constant d : unsigned(7 downto 0)) is
   begin
     print("Expecting: " & to_string(d));
     next_sample <= d;
     ready <= '1';
     wait until valid = '1';
     ready <= '0';
     assert data = std_logic_vector(d)
       report "Incorrect data received from DUT. Received: " & to_string(data) &
              ", Expected: " & to_string(d)  severity failure;
     print("    Test: Passed. ");
   end procedure;
```

```
procedure print(constant message : string) is
     variable str : line;
   begin
     write(str, message);
     writeline(output, str);
   end procedure;
```

Testbench contains BFM + DUT

Template provided to you



```
#
# Releasing reset
# Expecting: 11111111
#     Test: Passed.
# Expecting: 10000001
#     Test: Passed.
# Expecting: 00000000
#     Test: Passed.
# Expecting: 10000000
#     Test: Passed.
# Expecting: 00000001
#     Test: Passed.
# Expecting: 11110000
#     Test: Passed.
# Expecting: 10101010
#     Test: Passed.
# Expecting: 00001111
#     Test: Passed.
# Test: Completed
```

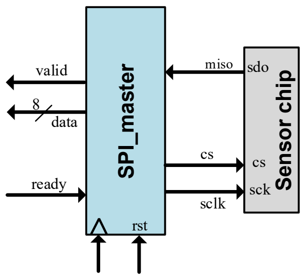
11

APPLICATION: AUTO-ADJUST BRIGHTNESS CONTROLLER

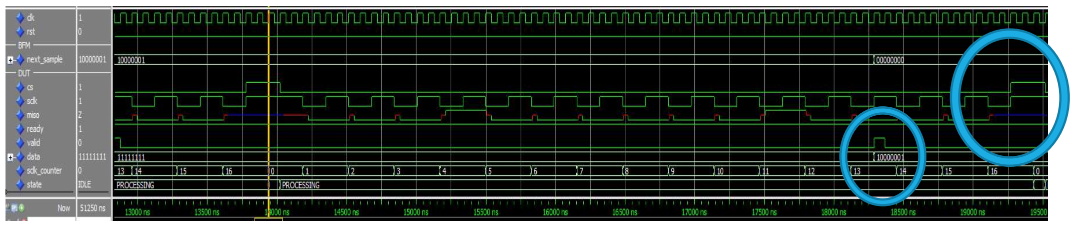
ENGINEERING DESIGN & DISCUSSION

12

PART 1: MASTER SPI CONTROLLER: CORE DESIGN & VERIFICATION

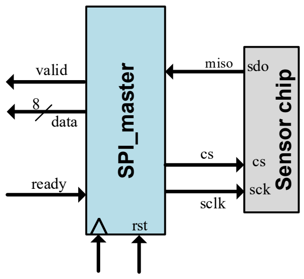


- When "ready" = 1 initiates SPI communication with light sensor chip (slave)
  - Master generates SCLK signals for slave chip + CS pulled low
- Create FSM with IDLE and TRANSMISSION states
  - IDLE: when ready = 1 + IDLE state -> TRANSMISSION
  - TRANSMISSION = initiate SCLK + CS = 0
    - obtain 8b serial **data according to datasheet**. Create 8b packet using shift register
    - Sample/Read on SCLK rising edge, data written by sensor on falling edge.
- Once serial 8x1b data obtained, send 8b packet on "data" + valid pulse on "valid"
  - Once 16 cycles complete -> Back to IDLE (set CS and SCLK back to '1' for IDLE)

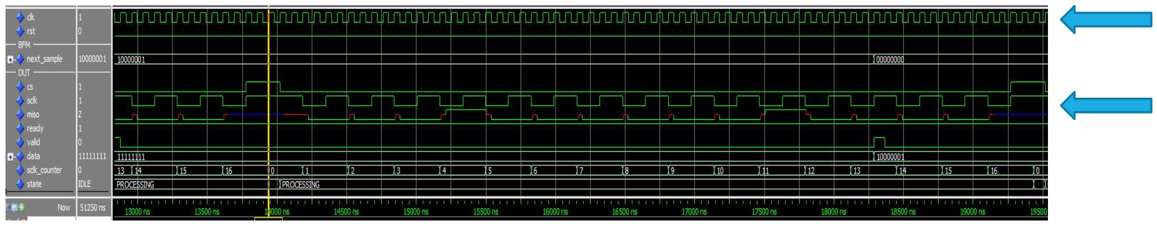


13

PART 1: MASTER SPI CONTROLLER: CORE DESIGN & VERIFICATION



- Counters for FSM
  - Must count 16 "sensor" cycles, but only has FPGA clock input
  - Prescaler for SCLK clock generation: 100 MHz -> 4MHz for chip
  - Use generic values so that design can be applied to any FPGA
    - Generate SCLK
  - Counter 0 (IDLE) 1 – 16 packets during TRANSMISSION. Ensure 16 cycles, but only 8 obtain actual data
- Synchronizer circuit for miso for metastability prevention
  - Will be the biggest issue, consider this when acquiring data (counter strategy may change)

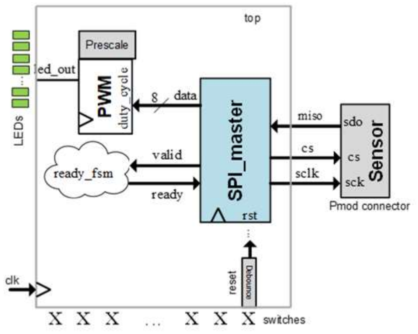


14

PART 2: AUTO-ADJUST BRIGHTNESS CONTROLLER

- **Objective:** Design a system that auto adjusts the brightness of the Zedboard's LEDs based on the ambient lighting conditions obtained from the sensor
  - The brighter the light shines on the sensor, the brighter the LEDs illuminate.
- Integrate the spi_controller and PWM controller from lab 1 to implement adaptive brightness
- Use the 8b data obtained from the ambient light sensor as the duty cycle to your PWM controller. PWM output will display on the onboard LEDs
  - 8bit resolution, no need to change original design.
- Include the prescaler circuit as input to the PWM controller to appropriately light and see the LED illumination

Top.vhd template provided to you

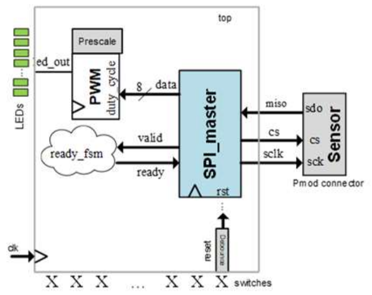


a) ready_fsm process is provided to you. Generates the ready signal for SPI to initiate a transaction.
b) Instantiate and connect the rest

15

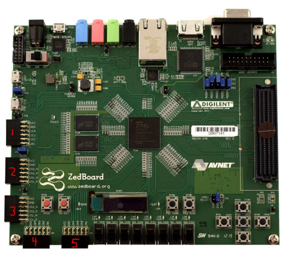
PART 2: AUTO-ADJUST BRIGHTNESS CONTROLLER

- Use a debouncer to debounce the reset. Output is applied to the rest of the circuit
- Create a testbench to verify that your circuit is operating as intended. Use the BFM and self-checking testbench to guide you.



Constraints File (xdc)

- Where to connect the sensor?
- Many pmod connectors on the Zedboard (5 to be exact). Select one and make the proper connections for CS, SCLK, and MISO (SDO).
  - Ensure the VCC and GND pins are connected 1:1
  - For no connections, leave unassigned/open
  - Use Zedboard-Master.xdc and Table 16 (Zedboard-UG18.pdf) to infer the proper pins for the pmod + LED + switches (refer to previous lab xdc and the Zedboard datasheets provided)



16