****

**Lab YP3**

**Light Sensor Integration**



**MIPSfpga 2.0. Lab YP3 - Integrating a peripheral: the light sensor example**

**1. Introduction**

In this lab you will review and synthesize a configuration of MIPSfpga system that contains a peripheral - Digilent Pmod ALS, the Ambient Light Sensor. In order to integrate a new peripheral into MIPSfpga system, you have to go through three main steps:

1. Design a Verilog module that handles the external protocol used to communicate to the peripheral. The protocol used in this lab is Serial Peripheral Interface (SPI).
2. Create glue logic used to interface the above module with AHB-Lite, on-chip bus fabric, used in MIPSfpga system.
3. Write software support that allows the application program running on MIPS microAptiv UP core inside MIPSfpga system to drive the peripheral using the corresponding memory-mapped input/output registers.

By going through this lab you will understand the fundamental difference between on-chip buses (AHB, AXI, OCP) and inter-chip buses (SPI, UART, I2C), as well as differences between serial buses and parallel buses. SPI bus used to communicate with the sensor is an example of a serial bus, while AHB-Lite used in MIPSfpga SoC is an example of a parallel bus.

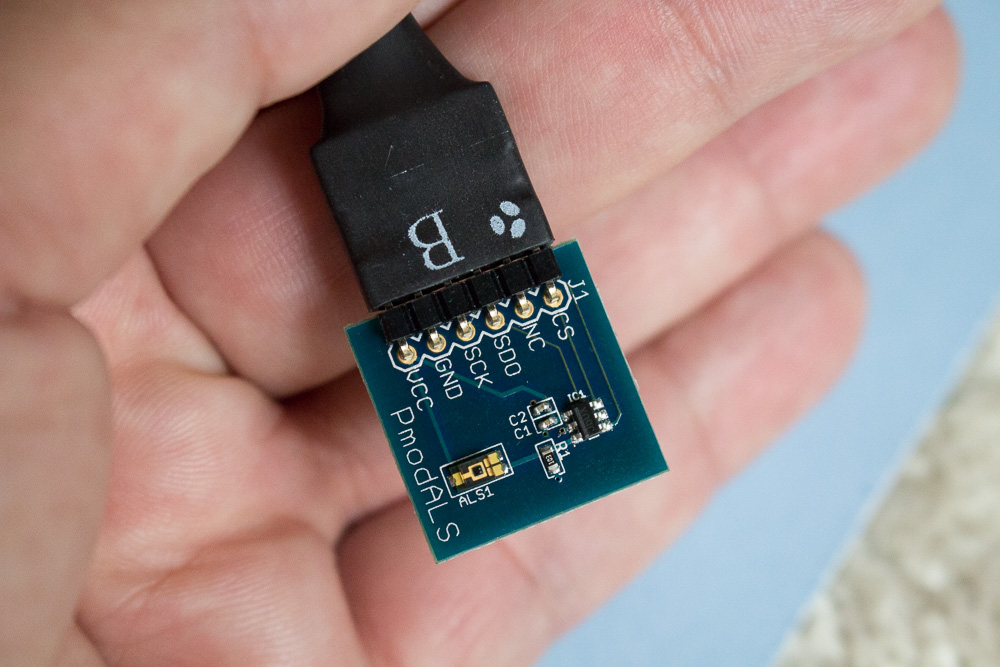
The result of light intensity, measured in this lab, is displayed on a multiple-digit seven-segment display. By combining a sensor, a system controller and an output device (the display) you will construct a practically useful gadget, a light meter.

This lab can be further combined with the next lab, *MIPSfpga 2.0. Lab YP4 - Introducing interrupts*, to demonstrate the interrupt-driven approach to input/output used in many real embedded systems.

**2. The theory of operation**

**Figure 1** shows the sensor used in this lab, Digilent PmodALS - Ambient Light Sensor. You can order this sensor from website [http://store.digilentinc.com/pmod-als-ambient-light-sensor](http://store.digilentinc.com/pmod-als-ambient-light-sensor/) for $9.99.

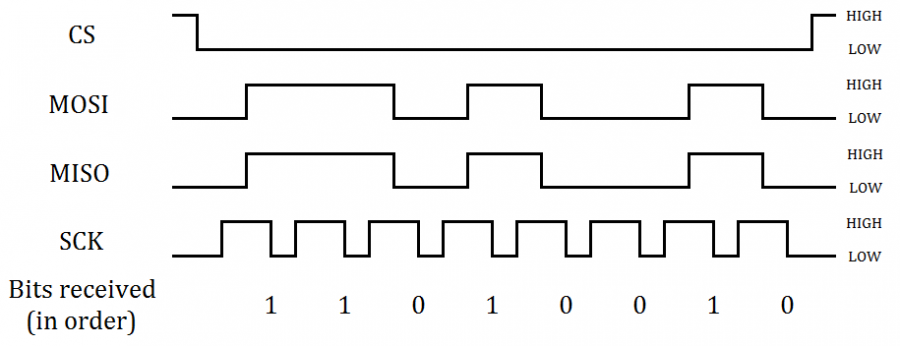
**Figure 1.**[**Digilent PmodALS - Ambient Light Sensor**](http://store.digilentinc.com/pmod-als-ambient-light-sensor/)

[](http://store.digilentinc.com/pmod-als-ambient-light-sensor/)

The sensor communicates with other devices using a protocol called Serial Peripheral Interface (SPI). This protocol is called serial because it transmits bits sequentially using few pins. The serial protocols are convenient to connect connect chips on a printed circuit boards, because the number of available pins coming out of a typical chip is limited.

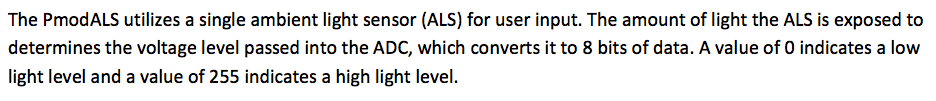
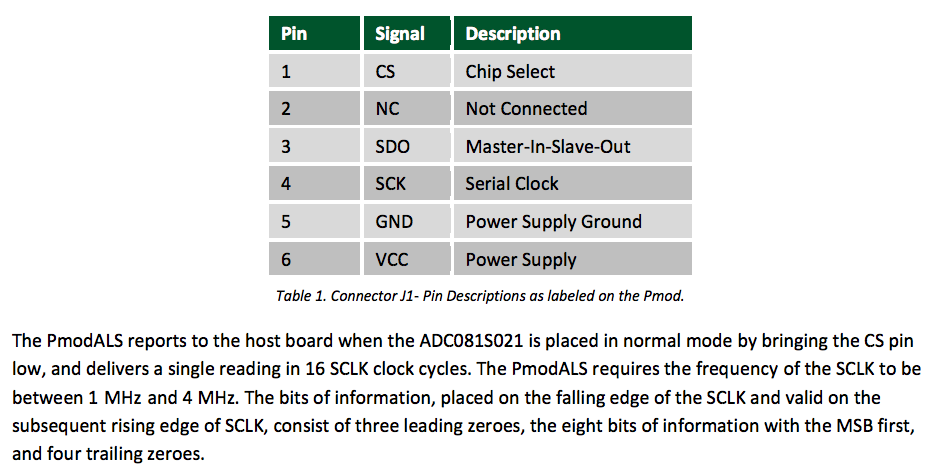
**Figure 2** illustrates how the information is transmitted using SPI protocol. You can get more information about the mechanics of SPI protocol from an article on Digilent web site at <https://reference.digilentinc.com/pmod:communication_protocols:spi>.

**Figure 2.**[**SPI protocol**](https://reference.digilentinc.com/pmod:communication_protocols:spi)**illustration from**[**Digilent**](http://digilentinc.com)**website**

[](https://reference.digilentinc.com/pmod:communication_protocols:spi)

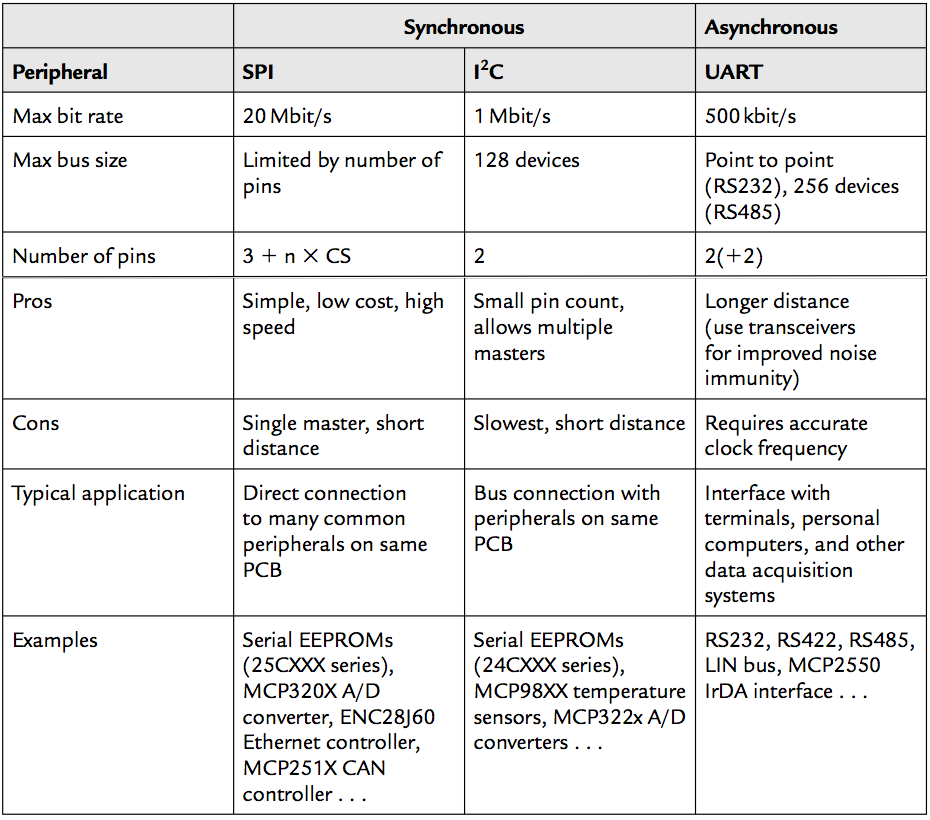
The specific variant of SPI protocol used by the light sensor is described in sensor documentation that can be downloaded from <https://reference.digilentinc.com/_media/reference/pmod/pmodals/pmodals_rm.pdf>. The exerpt from that documentation is on **Figure 3**.

**Figure 3. The description of a version of SPI protocol used in Digilent PmodALS - Ambient Light Sensor from**[**https://reference.digilentinc.com/\_media/reference/pmod/pmodals/pmodals\_rm.pdf**](https://reference.digilentinc.com/_media/reference/pmod/pmodals/pmodals_rm.pdf)

[[](https://reference.digilentinc.com/_media/reference/pmod/pmodals/pmodals_rm.pdf) [](https://reference.digilentinc.com/_media/reference/pmod/pmodals/pmodals_rm.pdf)](https://reference.digilentinc.com/_media/reference/pmod/pmodals/pmodals_rm.pdf)

SPI is not the only serial protocol that can be used to communicate with sensors, actuators and other computers. **Figure 4** contains a table that compares three most popular serial protocols used for simple point-to-point connections in embedded systems: SPI, UART and I2C.

**Figure 4. Serial protocol comparison table from a book**[**Programming 32-bit Microcontrollers in C: Exploring the PIC32 by Lucio Di Jasio**](https://www.amazon.com/Programming-32-bit-Microcontrollers-Exploring-Technology/dp/0750687096)

[](https://www.amazon.com/Programming-32-bit-Microcontrollers-Exploring-Technology/dp/0750687096)

Blocks inside systems on chips (SoCs) use different protocols to communicate with each other, including:

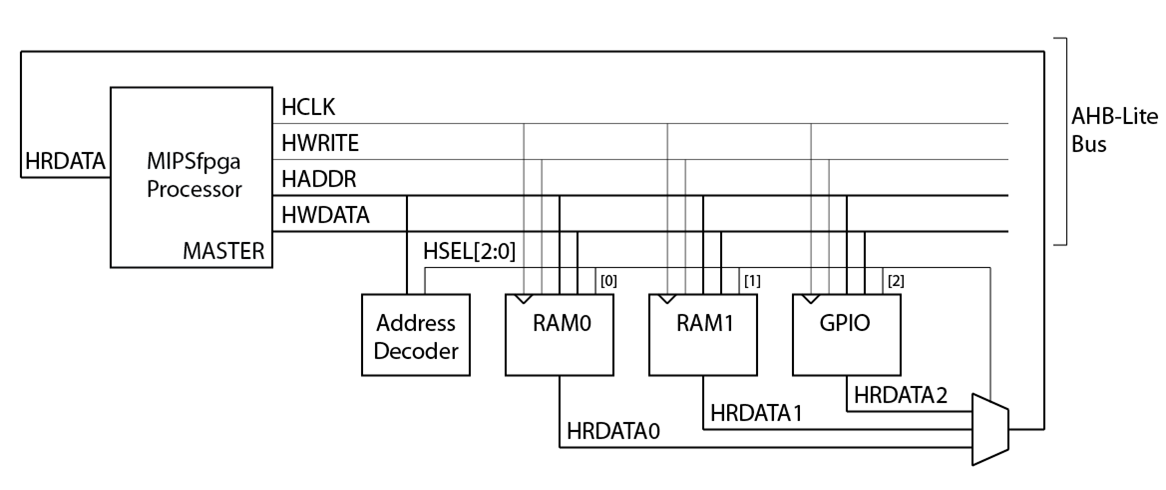
* Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI)
* AMBA Advanced High-performance Bus (AHB)
* Open Core Protocol (OCP)
* Processor Local Bus (PLB)
* Wishbone Bus and others

These protocols are parallel - they transmit multiple bits of information in one clock cycle, using multiple wires. Minimizing the number of wires for connectons inside a typical chip is not a critical task, more important is maximizing the amount of information transmitted per clock cycle.

In addition, synchronizing signals on multiple parallel wires inside the chip is much easier than outside. Outside the chip, noise and different wire length can be the issues. Because of it, the on-chip buses tend to be parallel, while off-chip protocols are frequently serial.

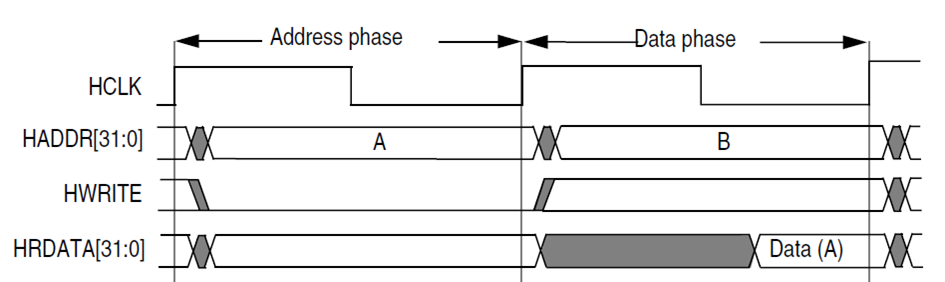
MIPS microAptiv UP core inside MIPSfpga SoC uses a protocol called AHB-Lite, a simplified variant of AHB, that assumes one master device and multiple slave devices in one system (full AHB allows multiple masters). **Figure 5** shows the general structure of MIPSfpga system based on AHB-Lite interconnect. The protocol is documented in *MIPS32® microAptiv™ UP Processor Core AHB-Lite Interface* manual included into MIPSfpga package.

**Figure 5. AHB-Lite interconnect in MIPSfpga system**

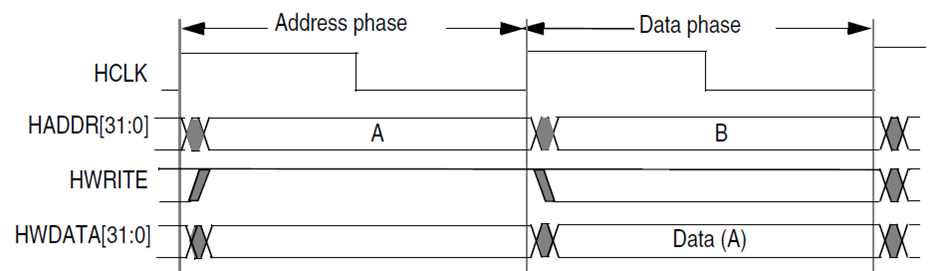


AHB-Lite transactions include single and burst variants of reads and writes. Address and data in those transactions are pipelined, which means that the address on a new transaction can be transmitted simultaneously with data for the previous transaction, as show on **Figure 6** for single reads and **Figure 7** for single writes.

**Figure 6. A waveform of a single AHB-Lite read transaction from**[**AHB-Lite specification**](http://mazsola.iit.uni-miskolc.hu/~drdani/docs_arm/IHI0033A_AMBA3_AHB_Lite.pdf)

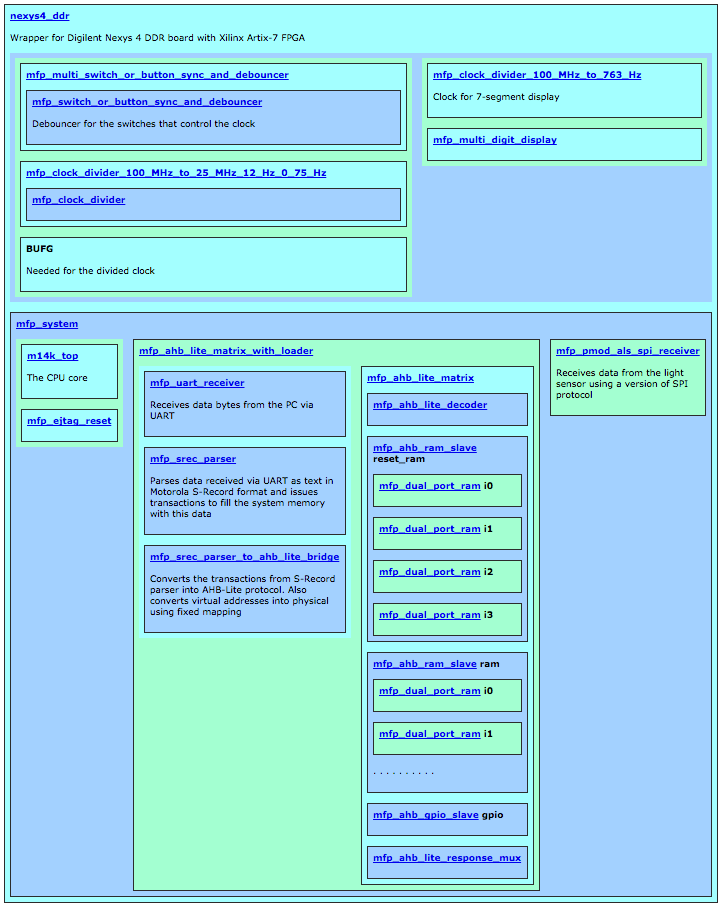
[](http://mazsola.iit.uni-miskolc.hu/~drdani/docs_arm/IHI0033A_AMBA3_AHB_Lite.pdf)

**Figure 7. A waveform of a single AHB-Lite write transaction from**[**AHB-Lite specification**](http://mazsola.iit.uni-miskolc.hu/~drdani/docs_arm/IHI0033A_AMBA3_AHB_Lite.pdf)

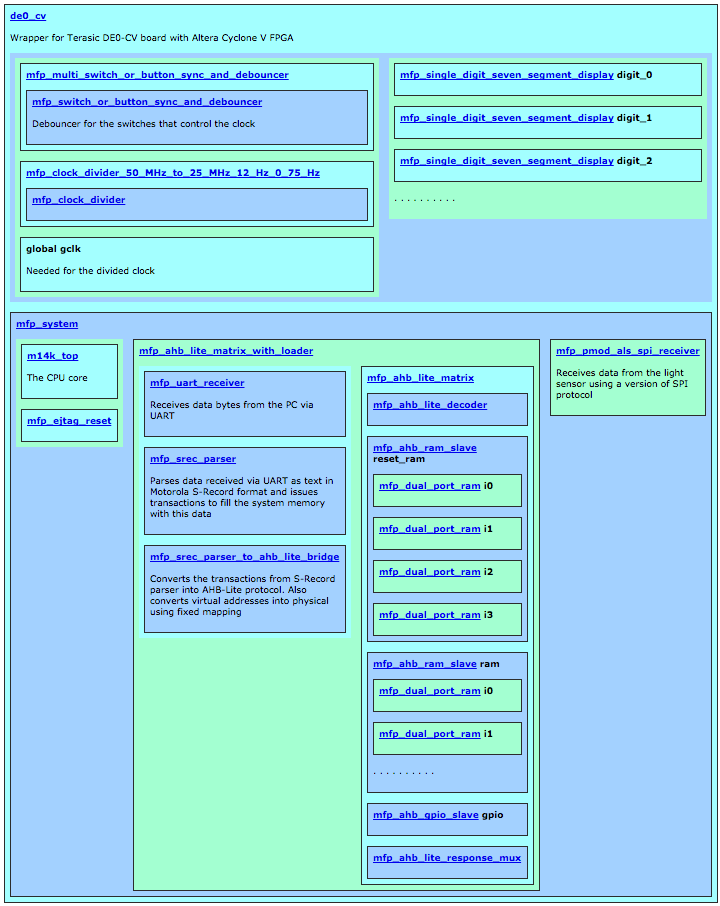
[](http://mazsola.iit.uni-miskolc.hu/~drdani/docs_arm/IHI0033A_AMBA3_AHB_Lite.pdf)

**Figure 8** shows how the light sensor module is instantiated in the module hierarchy for Digilent Nexys4 DDR board that carries Xilinx Artix-7 FPGA. **Figure 9** shows the same for Terasic DE0-CV board that carries Altera Cyclon V FPGA.

**Figure 8. MIPSfpga module hierarchy, including the light sensor module, for**[**Digilent Nexys4 DDR board**](http://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/)**that carries Xilinx Artix-7 FPGA**

[](http://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/)

**Figure 9. MIPSfpga module hierarchy, including the light sensor module, for**[**Terasic DE0-CVboard**](http://de0-cv.terasic.com)**that carries Altera Cyclon V FPGA**

[](http://de0-cv.terasic.com/)

**3. Lab steps**

This section outlines the sequence of steps, necessary to complete the lab. Almost all generic steps in this lab are the same as in *MIPSfpga 2.0 Lab YP1. Using MIPSfpga with Serial Loader Flow that does not require BusBlaster board and OpenOCD software*. Such generic steps are not described in this section. Only the steps different from *Lab YP1* are explained in details.

**3.1 Review the software part of the lab**

In order to understand what we are trying to achieve in this lab, it makes sense to start from the software. Review the following C program. The program reads a value from a memory-mapped I/O register that contains the current light sensor value. After reading the value, the program sends it to output devices: red and green LEDs and multiple-digit seven-segment display (if present on the board).

File *programs/03\_light\_sensor/main.c*

#include "mfp\_memory\_mapped\_registers.h"

int main ()

{

int n = 0;

for (;;)

{

MFP\_RED\_LEDS = MFP\_LIGHT\_SENSOR >> 4;

MFP\_7\_SEGMENT\_HEX = MFP\_LIGHT\_SENSOR;

MFP\_GREEN\_LEDS = MFP\_LIGHT\_SENSOR >> 4;

}

return 0;

}

Memory-mapped registers are defined in the following header file, included in the main program. As you can see, the address of the light-sensor I/O register is located in the uncached area of the memory with virtual address 0xBF800010 (physical address 0x1F800010). C *#define* macro *MFP\_LIGHT\_SENSOR* makes this register look just like a variable.

File *programs/03\_light\_sensor/mfp\_memory\_mapped\_registers.h*

#ifndef MFP\_MEMORY\_MAPPED\_REGISTERS\_H

#define MFP\_MEMORY\_MAPPED\_REGISTERS\_H

#define MFP\_RED\_LEDS\_ADDR 0xBF800000

#define MFP\_GREEN\_LEDS\_ADDR 0xBF800004

#define MFP\_SWITCHES\_ADDR 0xBF800008

#define MFP\_BUTTONS\_ADDR 0xBF80000C

#define MFP\_7\_SEGMENT\_HEX\_ADDR 0xBF800010

#define MFP\_LIGHT\_SENSOR\_ADDR 0xBF800014

#define MFP\_RED\_LEDS (\* (volatile unsigned \*) MFP\_RED\_LEDS\_ADDR )

#define MFP\_GREEN\_LEDS (\* (volatile unsigned \*) MFP\_GREEN\_LEDS\_ADDR )

#define MFP\_SWITCHES (\* (volatile unsigned \*) MFP\_SWITCHES\_ADDR )

#define MFP\_BUTTONS (\* (volatile unsigned \*) MFP\_BUTTONS\_ADDR )

#define MFP\_7\_SEGMENT\_HEX (\* (volatile unsigned \*) MFP\_7\_SEGMENT\_HEX\_ADDR )

#define MFP\_LIGHT\_SENSOR (\* (volatile unsigned \*) MFP\_LIGHT\_SENSOR\_ADDR )

**3.2 Review the hardware module that handles SPI protocol**

In this lab we don't need to handle all cases of SPI protocol. A generic flexible interface module would be quite long and complicated, such module can be licensed as a licensable IP core. However in this lab we are dealing with a specific sensor, and its interface is fixed: it simply produces 16 bits of data serially when *cs* ("chip select:) signal goes low. This specific version of SPI interface is also relatively slow, so we can sample the data simply by counting clock cycles and putting the received bits into a shift register on specific clock cycles.

Study the code below. How frequently does the signal *sample\_bit* go high? What about the signal *value\_done*? Can you explain or guess what would happen if we store the result in *value* more frequently?

File *system\_rtl/mfp\_pmod\_als\_spi\_receiver.v*

module mfp\_pmod\_als\_spi\_receiver

(

input clock,

input reset\_n,

output cs,

output sck,

input sdo,

output reg [15:0] value

);

reg [21:0] cnt;

reg [15:0] shift;

always @ (posedge clock or negedge reset\_n)

begin

if (! reset\_n)

cnt <= 22'b100;

else

cnt <= cnt + 22'b1;

end

assign sck = ~ cnt [3];

assign cs = cnt [8];

wire sample\_bit = ( cs == 1'b0 && cnt [3:0] == 4'b1111 );

wire value\_done = ( cnt [21:0] == 22'b0 );

always @ (posedge clock or negedge reset\_n)

begin

if (! reset\_n)

begin

shift <= 16'h0000;

value <= 16'h0000;

end

else if (sample\_bit)

begin

shift <= (shift << 1) | sdo;

end

else if (value\_done)

begin

value <= shift;

end

end

endmodule

**3.3 Review the glue logic that interface SPI interfacing module in AHB-Lite fabric**

Search for *MFP\_DEMO\_LIGHT\_SENSOR* symbol in *boards* and *system\_rtl* directories. Review the code fragments where that symbol occurs.

**3.3.1 System configuration**

Review and possibly modify the configuration parameters in the file *system\_rtl/mfp\_ahb\_lite\_matrix\_config.vh* as follows:

File *mfp\_ahb\_lite\_matrix\_config.vh*

//

// Configuration parameters

//

// `define MFP\_USE\_WORD\_MEMORY

// `define MFP\_INITIALIZE\_MEMORY\_FROM\_TXT\_FILE

// `define MFP\_USE\_SLOW\_CLOCK\_AND\_CLOCK\_MUX

`define MFP\_USE\_UART\_PROGRAM\_LOADER

`define MFP\_DEMO\_LIGHT\_SENSOR

// `define MFP\_DEMO\_INTERRUPTS

// `define MFP\_DEMO\_CACHE\_MISSES

// `define MFP\_DEMO\_PIPE\_BYPASS

. . . . . . . . . . . . . . . . . . . . . .

Review the added `defines for the physical address of the memory-mapped register and I/O identification number for the added light sensor peripheral in the same file *system\_rtl/mfp\_ahb\_lite\_matrix\_config.vh*:

File *system\_rtl/mfp\_ahb\_lite\_matrix\_config.vh*

. . . . . . . . . . . . . . . . . . . . . .

`define MFP\_RED\_LEDS\_ADDR 32'h1f800000

`define MFP\_GREEN\_LEDS\_ADDR 32'h1f800004

`define MFP\_SWITCHES\_ADDR 32'h1f800008

`define MFP\_BUTTONS\_ADDR 32'h1f80000C

`define MFP\_7\_SEGMENT\_HEX\_ADDR 32'h1f800010

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

`define MFP\_LIGHT\_SENSOR\_ADDR 32'h1f800014

`endif

`define MFP\_RED\_LEDS\_IONUM 4'h0

`define MFP\_GREEN\_LEDS\_IONUM 4'h1

`define MFP\_SWITCHES\_IONUM 4'h2

`define MFP\_BUTTONS\_IONUM 4'h3

`define MFP\_7\_SEGMENT\_HEX\_IONUM 4'h4

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

`define MFP\_LIGHT\_SENSOR\_IONUM 4'h5

`endif

**3.3.2 Review the board wrapper file for Xilinx**

The top-level module *nexys4\_ddr* instantiates a board-independent module *mfp\_system* and connect the designated GPIO pins to SPI inputs of *mfp\_system* module:

File *boards/nexys4\_ddr/nexys4\_ddr.v*

module nexys4\_ddr

(

input CLK100MHZ,

input CPU\_RESETN,

. . . . . . . . . . . . . . . . . . . .

inout [12:1] JA,

inout [12:1] JB,

. . . . . . . . . . . . . . . . . . . .

);

. . . . . . . . . . . . . . . . . . . .

mfp\_system mfp\_system

(

.SI\_ClkIn ( clk ),

.SI\_Reset ( ~ CPU\_RESETN ),

. . . . . . . . . . . . . . . . . .

.SPI\_CS ( JA [ 1] ),

.SPI\_SCK ( JA [ 4] ),

.SPI\_SDO ( JA [ 3] )

);

assign JA [7] = 1'b0;

**3.3.3 Review the board wrapper file for Altera**

The top-level module *de0\_cv* instantiates a board-independent module *mfp\_system* and connect the designated GPIO pins to SPI inputs of *mfp\_system* module:

File *boards/de0\_cv/de0\_cv.v*

module de0\_cv

(

input CLOCK2\_50,

input CLOCK3\_50,

inout CLOCK4\_50,

input CLOCK\_50,

input RESET\_N,

. . . . . . . . . . . . . . . . . . . .

inout [35:0] GPIO\_0,

inout [35:0] GPIO\_1

);

. . . . . . . . . . . . . . . . . . . .

mfp\_system mfp\_system

(

.SI\_ClkIn ( clk ),

.SI\_Reset ( ~ RESET\_N ),

. . . . . . . . . . . . . . . . . .

.SPI\_CS ( GPIO\_1 [34] ),

.SPI\_SCK ( GPIO\_1 [28] ),

.SPI\_SDO ( GPIO\_1 [30] )

);

. . . . . . . . . . . . . . . . . . . .

assign GPIO\_1 [26] = 1'b0;

**3.3.4 Review the board-independent top system module**

Note this module instantiates the CPU core, the AHB-Lite interconnect and the newly added SPI interfacing module that works with the light sensor:

File *system\_rtl/mfp\_system.v*

module mfp\_system

(

input SI\_ClkIn,

input SI\_ColdReset,

input SI\_Reset,

. . . . . . . . . . . . . . . . . . . .

output SPI\_CS,

output SPI\_SCK,

input SPI\_SDO

);

. . . . . . . . . . . . . . . . . . . .

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

wire [15:0] IO\_LightSensor;

`endif

mfp\_ahb\_lite\_matrix\_with\_loader ahb\_lite\_matrix

(

.HCLK ( HCLK ),

.HRESETn ( ~ SI\_Reset ), // Not HRESETn - this is necessary for serial loader

. . . . . . . . . . . . . . . . . .

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

.IO\_LightSensor ( IO\_LightSensor ),

`endif

. . . . . . . . . . . . . . . . . .

);

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

mfp\_pmod\_als\_spi\_receiver mfp\_pmod\_als\_spi\_receiver

(

.clock ( SI\_ClkIn ),

.reset\_n ( ~ SI\_Reset ),

.cs ( SPI\_CS ),

.sck ( SPI\_SCK ),

.sdo ( SPI\_SDO ),

.value ( IO\_LightSensor )

);

`endif

**3.3.5 Review the code that propagates the received light sensor value down the module hierarchy**

File *system\_rtl/mfp\_ahb\_lite\_matrix\_with\_loader.v*

module mfp\_ahb\_lite\_matrix\_with\_loader

(

input HCLK,

input HRESETn,

. . . . . . . . . . . . . . . . . . . .

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

input [15:0] IO\_LightSensor,

`endif

. . . . . . . . . . . . . . . . . . . .

);

. . . . . . . . . . . . . . . . . . . .

mfp\_ahb\_lite\_matrix ahb\_lite\_matrix

(

.HCLK ( HCLK ),

.HRESETn ( HRESETn ),

. . . . . . . . . . . . . . . . . .

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

.IO\_LightSensor ( IO\_LightSensor ),

`endif

. . . . . . . . . . . . . . . . . .

);

File *system\_rtl/mfp\_ahb\_lite\_matrix.v*

module mfp\_ahb\_lite\_matrix

(

input HCLK,

input HRESETn,

. . . . . . . . . . . . . . . . . . . .

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

input [15:0] IO\_LightSensor,

`endif

. . . . . . . . . . . . . . . . . . . .

);

. . . . . . . . . . . . . . . . . . . .

mfp\_ahb\_gpio\_slave gpio

(

.HCLK ( HCLK ),

.HRESETn ( HRESETn ),

. . . . . . . . . . . . . . . . . .

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

,

.IO\_LightSensor ( IO\_LightSensor )

`endif

);

**3.3.6 Review how GPIO slave connects the received value to the system bus**

The general-purpose input-output module connects the wires coming from several peripherals to AHB-Lite system bus in order to make these peripherals visible to the software. The peripherals include buttons, switches, LEDs and now the light sensor:

File *mfp\_ahb\_gpio\_slave.v*

`include "mfp\_ahb\_lite.vh"

`include "mfp\_ahb\_lite\_matrix\_config.vh"

module mfp\_ahb\_gpio\_slave

(

input HCLK,

input HRESETn,

input [31:0] HADDR,

input [ 2:0] HBURST,

input HMASTLOCK,

input [ 3:0] HPROT,

input [ 2:0] HSIZE,

input HSEL,

input [ 1:0] HTRANS,

input [31:0] HWDATA,

input HWRITE,

output reg [31:0] HRDATA,

output HREADY,

output HRESP,

input SI\_Endian,

input [`MFP\_N\_SWITCHES - 1:0] IO\_Switches,

input [`MFP\_N\_BUTTONS - 1:0] IO\_Buttons,

output reg [`MFP\_N\_RED\_LEDS - 1:0] IO\_RedLEDs,

output reg [`MFP\_N\_GREEN\_LEDS - 1:0] IO\_GreenLEDs,

output reg [`MFP\_7\_SEGMENT\_HEX\_WIDTH - 1:0] IO\_7\_SegmentHEX

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

,

input [15:0] IO\_LightSensor

`endif

);

// Ignored: HMASTLOCK, HPROT

// TODO: SI\_Endian

// Assignments to HREADY and HTRANS should be modified

// for more complicated peripherals

assign HREADY = 1'b1;

assign HRESP = 1'b0;

reg [ 1:0] HTRANS\_dly;

reg [31:0] HADDR\_dly;

reg HWRITE\_dly;

reg HSEL\_dly;

always @ (posedge HCLK)

begin

HTRANS\_dly <= HTRANS;

HADDR\_dly <= HADDR;

HWRITE\_dly <= HWRITE;

HSEL\_dly <= HSEL;

end

wire [3:0] read\_ionum = HADDR [5:2];

wire [3:0] write\_ionum = HADDR\_dly [5:2];

wire write\_enable = HTRANS\_dly != `HTRANS\_IDLE && HSEL\_dly && HWRITE\_dly;

always @ (posedge HCLK or negedge HRESETn)

begin

if (! HRESETn)

begin

IO\_RedLEDs <= `MFP\_N\_RED\_LEDS'b0;

IO\_GreenLEDs <= `MFP\_N\_GREEN\_LEDS'b0;

IO\_7\_SegmentHEX <= `MFP\_7\_SEGMENT\_HEX\_WIDTH'b0;

end

else if (write\_enable)

begin

case (write\_ionum)

`MFP\_RED\_LEDS\_IONUM : IO\_RedLEDs <= HWDATA [`MFP\_N\_RED\_LEDS - 1:0];

`MFP\_GREEN\_LEDS\_IONUM : IO\_GreenLEDs <= HWDATA [`MFP\_N\_GREEN\_LEDS - 1:0];

`MFP\_7\_SEGMENT\_HEX\_IONUM : IO\_7\_SegmentHEX <= HWDATA [`MFP\_7\_SEGMENT\_HEX\_WIDTH - 1:0];

endcase

end

end

always @ (posedge HCLK or negedge HRESETn)

begin

if (! HRESETn)

begin

HRDATA <= 32'h00000000;

end

else

begin

case (read\_ionum)

`MFP\_SWITCHES\_IONUM : HRDATA <= { { 32 - `MFP\_N\_SWITCHES { 1'b0 } } , IO\_Switches };

`MFP\_BUTTONS\_IONUM : HRDATA <= { { 32 - `MFP\_N\_BUTTONS { 1'b0 } } , IO\_Buttons };

`ifdef MFP\_DEMO\_LIGHT\_SENSOR

`MFP\_LIGHT\_SENSOR\_IONUM : HRDATA <= { 16'b0, IO\_LightSensor };

`endif

default: HRDATA <= 32'h00000000;

endcase

end

end

endmodule

**3.5. Connect the light sensor to the board**

For *Digilent* boards, such as *Nexys4*, *Nexys4 DDR* or *Basys3*, the light sensor can be just inserted into the proper position of JA or JB port. Please see the pin information in .XDC file and in the board documentation from Digilent to figure out how to connect the sensor to Digulent boards. For *Altera/Terasic* boards you need to use female-to-female jumper wires to connect the sensor to the appropriate GPIO pins.

**3.6. Connect the board to the computer**

For *Digilent* boards, such as *Nexys4*, *Nexys4 DDR* or *Basys3*, this step is obvious. For *Altera/Terasic* boards some additional steps required:

1. Connect USB-to-UART connector to FPGA board. Either *FT232RL* or *PL2303TA* that you can by from AliExpress or Radio Shack will do the job. *TX* output from the connector (green wire on *PL2303TA*) should go to pin 3 from right bottom on Terasic DE0, DE0-CV, DE1, DE2-115 (right top on DE0-Nano) and *GND* output (black wire on *PL2303TA*) should be connected to pin 6 from right bottom on Terasic DE0, DE0-CV, DE1, DE2-115 (right top on DE0-Nano). Please consult photo picture in *Lab YP1* to avoid short-circuit or other connection problems.
2. For *FT232RL* connector: make sure to set 3.3V/5V jumper on *FT232RL* part to 3.3V.
3. For the boards that require external power in addition to the power that comes from USB, connect the power supply. The boards that require the extra power supply include *Terasic DE2-115*.
4. Connect FPGA board to the computer using main connection cable provided by the board manufacturers. Make sure to put USB cable to the right jack when ambiguity exists (such as in *Terasic DE2-115* board).
5. Make sure to power the FPGA board (turn on the power switch) before connecting the UART cable from USB-to-UART connector to the computer. Failing to do so may result in electric damage to the board.
6. Connect USB-to-UART connector to FPGA board.

**3.6 Run the synthesis and configure the FPGA with the synthesized MIPSfpga system**

This step is identical to the synthesis step in *Lab YP1*

**3.7 Go to the lab directory and clean it up**

Under Windows:

cd programs\lab\_yp2

00\_clean\_all.bat

Under Linux:

cd programs/lab\_yp2

00\_clean\_all.sh

**3.8 Prepare the first software run**

Following the procedure described in *Lab YP1*, compile and link the program, generate Motorola S-Record file and upload this file into the memory of the synthesized MIPSfpga-based system on the board.

Under Windows:

1. cd programs\lab\_yp2
2. run 02\_compile\_and\_link.bat
3. run 08\_generate\_motorola\_s\_record\_file.bat
4. run 11\_check\_which\_com\_port\_is\_used.bat
5. edit 12\_upload\_to\_the\_board\_using\_uart.bat based on the result from the previous step - set the working port in "set a=" assignment.
6. Make sure the switches 0 and 1 on FPGA board are turned off. Switches 0 and 1 control the speed of the clock. If the switches 0 and 1 are not off, the loading through UART is not going to work.
7. run 12\_upload\_to\_the\_board\_using\_uart.bat

Under Linux:

If uploading program to the board first time during the current Linux session, add the current user to *dialout* Linux group. Enter the *root* password when prompted:

sudo adduser $USER dialout

su - $USER

After that:

1. cd programs/lab\_yp2
2. run ./02\_compile\_and\_link.sh
3. run ./08\_generate\_motorola\_s\_record\_file.sh
4. run ./11\_check\_which\_com\_port\_is\_used.sh
5. edit ./12\_upload\_to\_the\_board\_using\_uart.sh based on the result from the previous step - set the working port in â\_oset a=â\_\_ assignment
6. Make sure the switches 0 and 1 on FPGA board are turned off. Switches 0 and 1 control the speed of the clock. If the switches 0 and 1 are not off, the loading through UART is not going to work.
7. ./run 12\_upload\_to\_the\_board\_using\_uart.sh

**3.9 Run the software on the board**

Reset the processor. The reset buttons for each board are listed in the table below:

|  |  |
| --- | --- |
| **Board** | **Reset button** |
| Digilent Basys3 | Up |
| Digilent Nexys4 | Dedicated CPU Reset |
| Digilent Nexys4 DDR | Dedicated CPU Reset |
| Terasic DE0 | Button/Key 0 |
| Terasic DE0-CV | Dedicated reset button |
| Terasic DE0-Nano | Button/Key 0 |
| Terasic DE1 | Button/Key 0 |
| Terasic DE2-115 | Button/Key 0 |
| Terasic DE10-Lite | Button/Key 0 |

At this moment you should see the light meter working - the seven-segment display should start displaying the level of light. If it is not working, check the cables, software and hardware configuration. Try bringing the board to the source of light, then cover it with your hand and watch the result on the display going to 0.

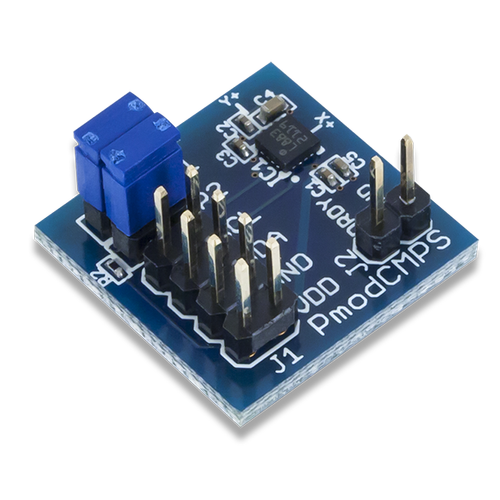
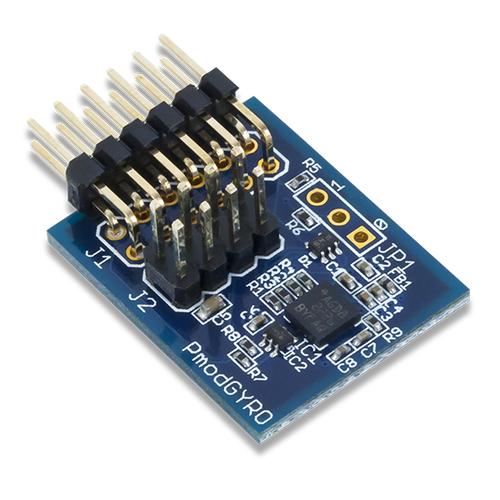
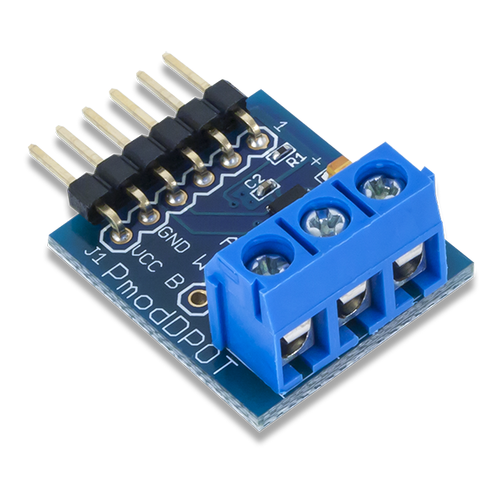
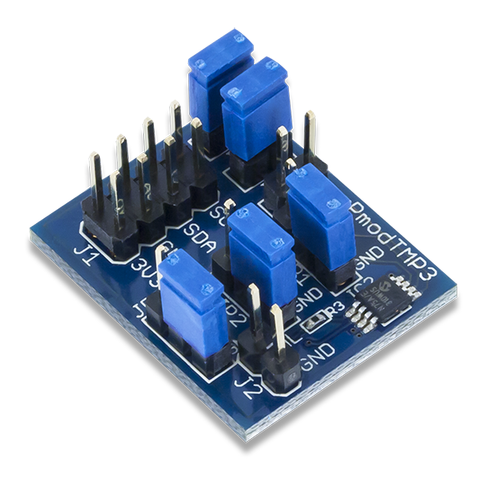
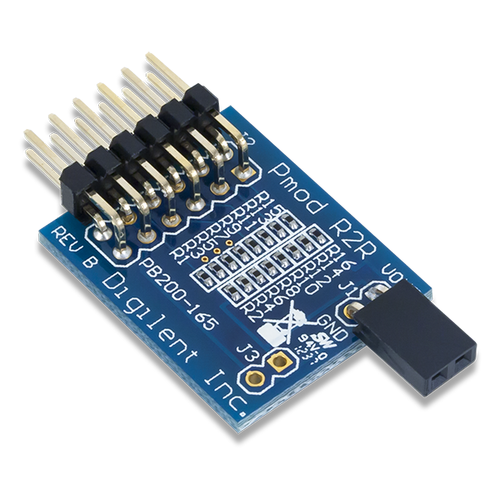
**4. Follow-up projects and exercises**

In a real embedded system, the input-output is frequently interrupt-driven. Instead of constantly polling memory-mapped input-output registers, the software performs more important tasks, such as computations. The input-output actions happen only when the peripheral device sends an interrupt request.

After going through the next lab (*MIPSfpga 2.0. Lab YP4 - Introducing interrupts*), come back to this lab and modify the light sensor interfacing module. The modified module should issue an interrupt when the measured value changes. Connect the interrupt pin to *SI\_Int* signal of MIPS microAptive UP core. Measure the system performance improvement that comes from offloading input-output to the interrupt service routine.

You can use the light sensor lab and the interrupt lab as examples to develop multiple projects, integrating sensors and actuators into MIPSfpga system. Digilent, a National Instruments company, offers a number of peripheral modules that can be relatively easily integrated with MIPSfpga. These modules can be ordered from <http://store.digilentinc.com/pmod-modules>.

**Figure 10. Various**[**peripheral modules from Digilent**](http://store.digilentinc.com/pmod-modules/)**that can be relatively easily integrated with MIPSfpga**

[[](http://store.digilentinc.com/pmod-modules/)   
[](http://store.digilentinc.com/pmod-modules/)   
[](http://store.digilentinc.com/pmod-modules/)](http://store.digilentinc.com/pmod-modules/)