MIPS Open Developer Day Package. Lab MO3 - Integrating a peripheral: the light sensor example

0. Note. This documentation is in the process of being updated

This documentation is in the process of being updated to match the latest version of the source code. The planned updates include:

1. Informing the user how to use makefiles instead of short Linux scripts, Windows batch files and GUI. To see all available make actions just run *make* in any board or program directory example:

   make in subdirectories of *boards* directory:

   ```
   make help   - show this message
   make all    - clean, create the board project and run the synthesis
   make clean  - delete synth folder
   make create - create the board project
   make open   - open the board project
   make build  - build the board project
   make load   - program the FPGA board
   ```

   make in subdirectories of *programs* directory:

   ```
   make help       - show this message
   make all        - alternative for: compile program size disasm readmemh srecord
   make program    - build program.elf from sources
   make compile    - compile all C sources to ASM
   make size       - show program size information
   make disasm     - disassemble program.elf
   make readmemh   - create verilog memory init file for simulation
   make srecord    - create Motorola S-record file to use it with UART loader
   make clean      - delete all created files
   make load       - load program into the device memory, run it and detach gdb
   make debug      - load program into the device memory, wait for gdb commands
   make attach     - attach to the device, wait for gdb commands
   make uart       - load program into the device memory using UART loader
   make modelsim   - simulate program and device using Modelsim
   make icarus     - simulate program and device using Icarus Verilog
   make gtkwave    - show the result of Icarus Verilog simulation in GTKWave
   ```

2. Adding support for *make uart UART=N* where *N* is USB-to-UART device number (*/dev/ttyUSB0, 1, 2, ... N*).

3. Light Sensor in peripheral integration lab is now integrated not as an addional GPIO (General Purpose I/O) device inside GPIO AHB-Lite slave, but as a separate AHB-Lite slave.

4. Source code for pipeline bypass lab (*Lab MO6 - The first glance into pipelining*) does not match the documentation. Please look inside the code to see what it does.

5. The new CorExtend / UDI - User Defined Instructions example does not have an instruction. Please refer to MIPS Open Day slides to figure out what it does.

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, I^2C), 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.

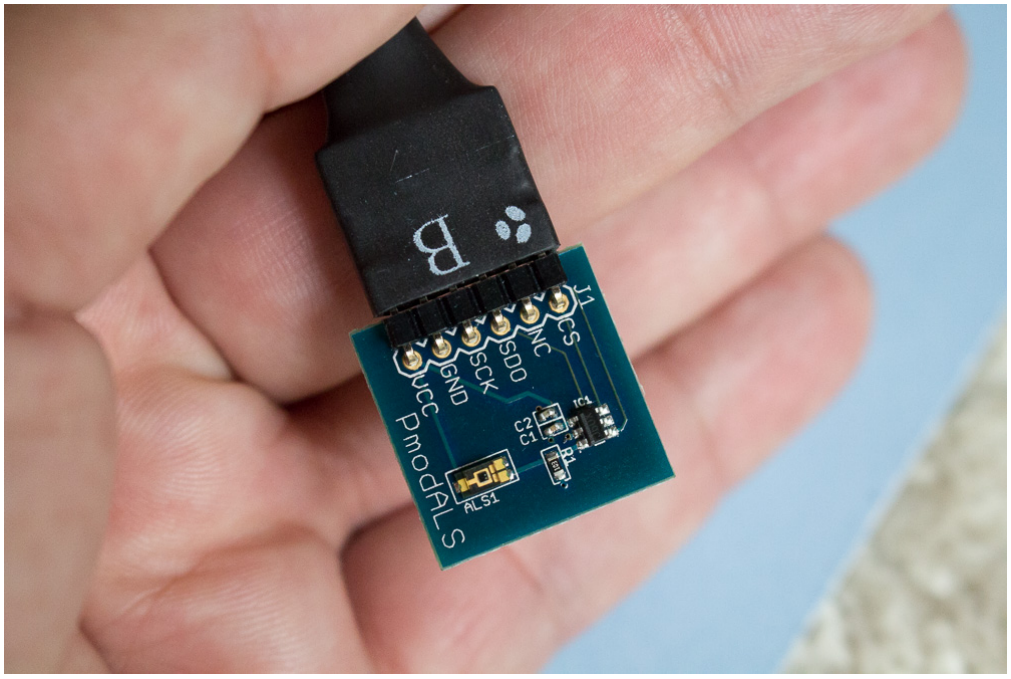
The result of light intensity, measured in this lab, is displayed on a multiple-digit seven-segment display. By combining a sensor, a system controller and an output device (the display) you will construct a practically useful gadget, a light meter.

This lab can be further combined with the next lab, *MIPSfpga. Lab MO4 - 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 for $9.99.

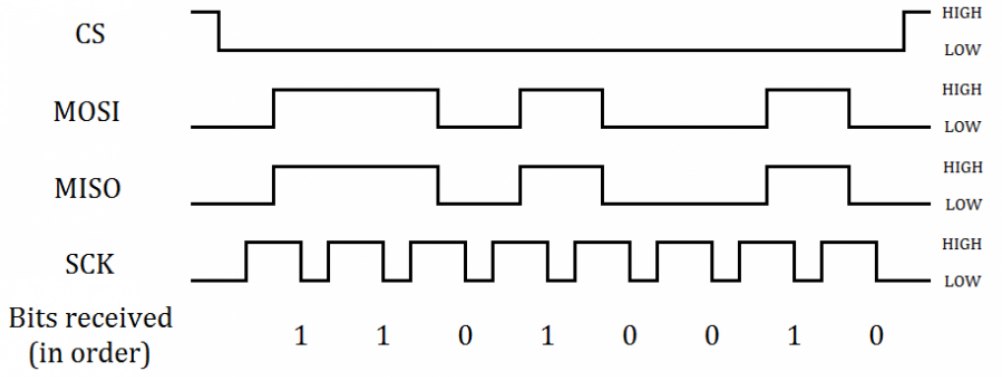
Figure 1. Digilent PmodALS - Ambient Light Sensor



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 illustration from Digilent website



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

The PmodALS utilizes a single ambient light sensor (ALS) for user input. The amount of light the ALS is exposed to determines the voltage level passed into the ADC, which converts it to 8 bits of data. A value of 0 indicates a low light level and a value of 255 indicates a high light level.

Pin	Signal	Description
1	CS	Chip Select
2	NC	Not Connected
3	SDO	Master-In-Slave-Out
4	SCK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply

Table 1. Connector J1- Pin Descriptions as labeled on the Pmod.

The PmodALS reports to the host board when the ADC081S021 is placed in normal mode by bringing the CS pin low, and delivers a single reading in 16 SCLK clock cycles. The PmodALS requires the frequency of the SCLK to be between 1 MHz and 4 MHz. The bits of information, placed on the falling edge of the SCLK and valid on the subsequent rising edge of SCLK, consist of three leading zeroes, the eight bits of information with the MSB first, and four trailing zeroes.

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 I^2C.

Figure 4. Serial protocol comparison table from a book Programming 32-bit Microcontrollers in C: Exploring the PIC32 by Lucio Di Jasio

	Synchronous		Asynchronous
Peripheral	**SPI**	**I^2C**	**UART**
Max bit rate	20 Mbit/s	1 Mbit/s	500 kbit/s
Max bus size	Limited by number of pins	128 devices	Point to point (RS232), 256 devices (RS485)
Number of pins	3 + n × CS	2	2(+2)
Pros	Simple, low cost, high speed	Small pin count, allows multiple masters	Longer distance (use transceivers for improved noise immunity)
Cons	Single master, short distance	Slowest, short distance	Requires accurate clock frequency
Typical application	Direct connection to many common peripherals on same PCB	Bus connection with peripherals on same PCB	Interface with terminals, personal computers, and other data acquisition systems
Examples	Serial EEPROMs (25CXXX series), MCP320X A/D converter, ENC28J60 Ethernet controller, MCP251X CAN controller . . .	Serial EEPROMs (24CXXX series), MCP98XX temperature sensors, MCP322x A/D converters . . .	RS232, RS422, RS485, LIN bus, MCP2550 IrDA interface . . .

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

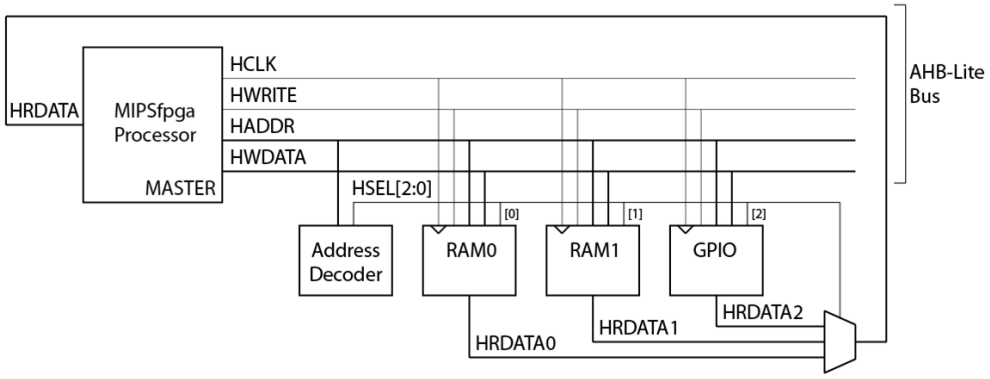
- Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI)
- AMBA Advanced High-performance Bus (AHB)
- Open Core Protocol (OCP)
- Processor Local Bus (PLB)
- Wishbone Bus and others

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

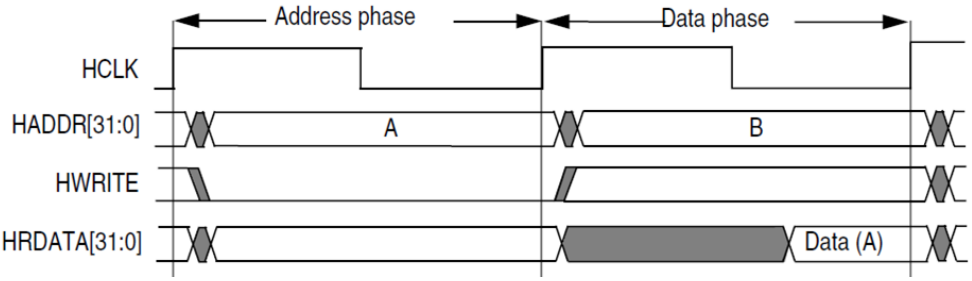


Figure 7. A waveform of a single AHB-Lite write transaction from AHB-Lite specification

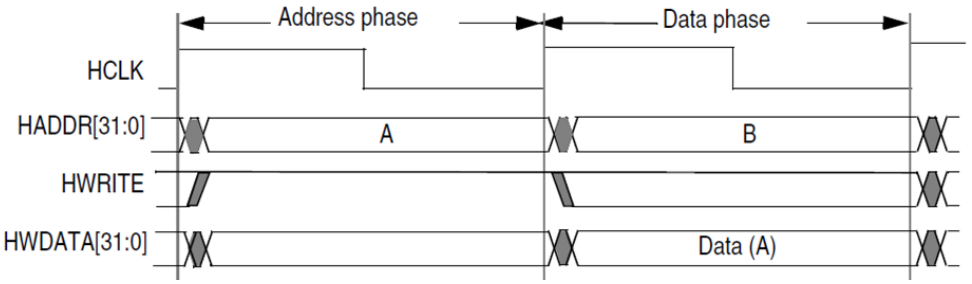


Figure 8 shows how the light sensor module is instantiated in the module hierarchy for Digilent Nexys4 DDR board that carries Xilinx Artix-7 FPGA. **Figure 9** shows the same for Terasic DE0-CV board that carries Altera Cyclon V FPGA.

Figure 8. MIPSfpga module hierarchy, including the light sensor module, for Digilent Nexys4 DDR board that carries Xilinx Artix-7 FPGA

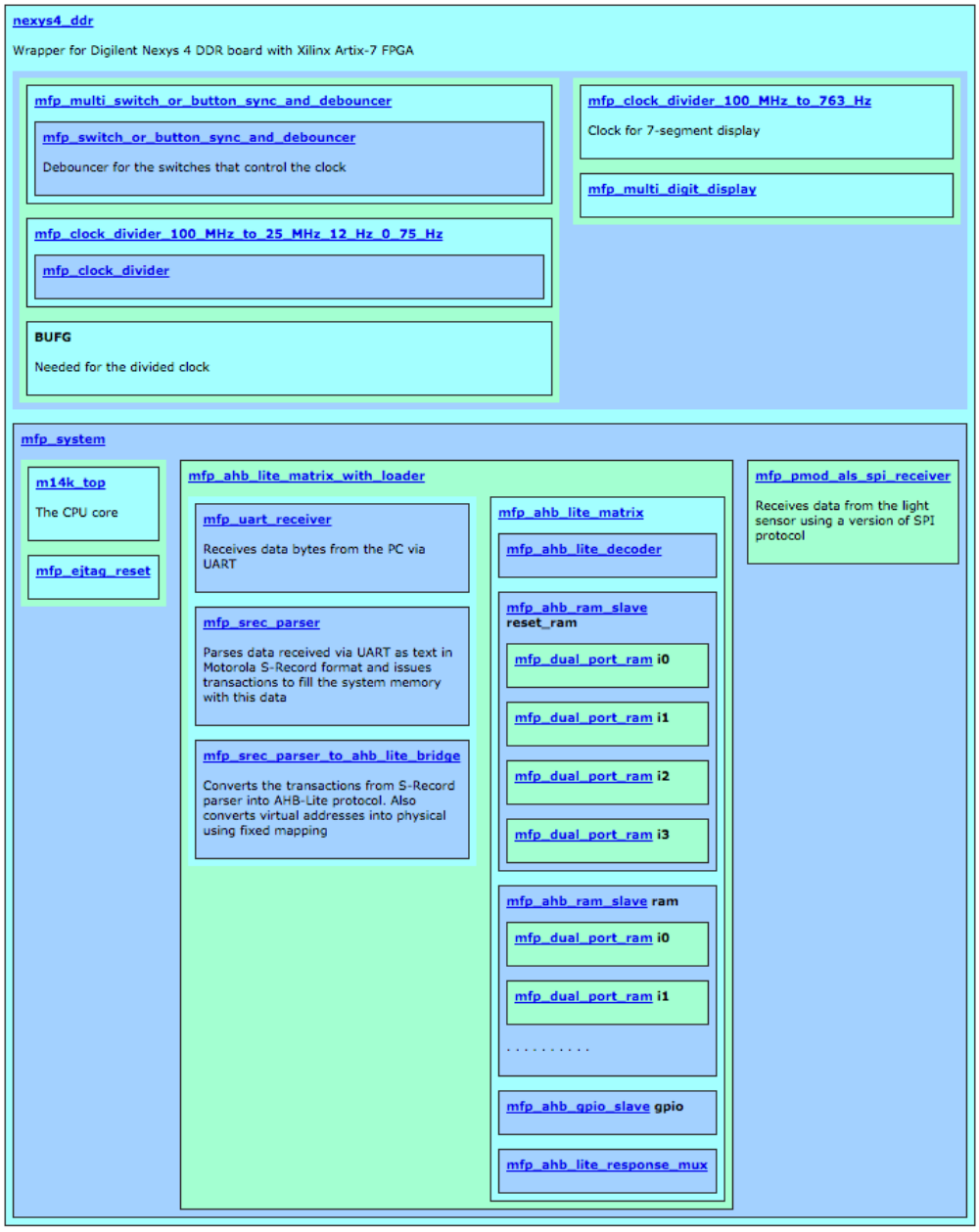
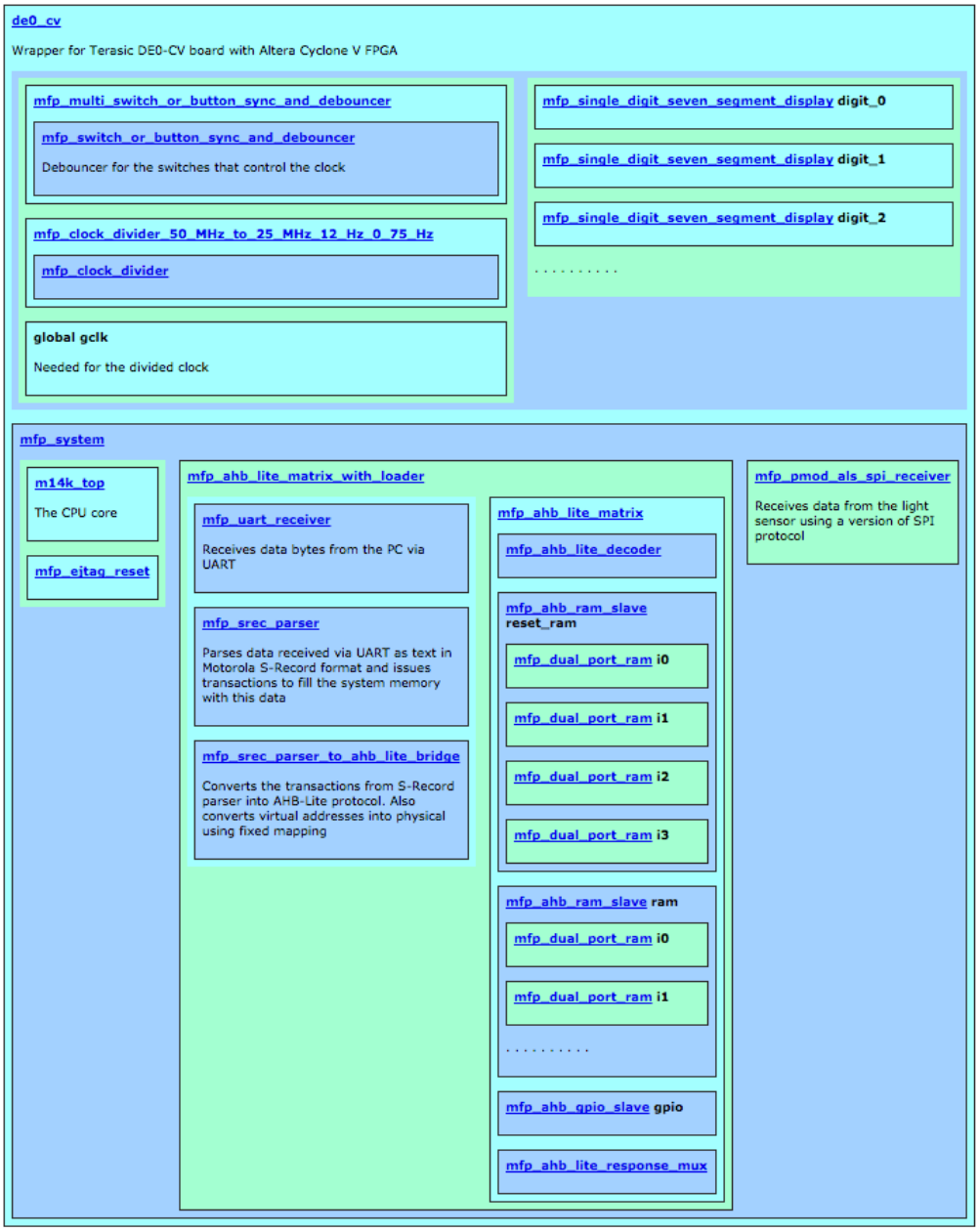


Figure 9. MIPSfpga module hierarchy, including the light sensor module, for Terasic DE0-CVboard that carries Altera Cyclon V FPGA



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 MO1. 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 MO1* 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

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

        `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_Switch
                `MFP_BUTTONS_IONUM       : HRDATA <= { { 32 - `MFP_N_BUTTONS  { 1'b0 } } , IO_Button

                `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 MO1* 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 MO1*

3.7 Go to the lab directory and clean it up

Under Windows:

```
cd programs\lab_mo2
00_clean_all.bat
```

Under Linux:

```
cd programs/lab_mo2
00_clean_all.sh
```

3.8 Prepare the first software run

Following the procedure described in *Lab MO1*, 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_mo2
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_mo2
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. Lab MO4 - 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 that can be relatively easily integrated with MIPSfpga