MIPSfpga
by Imagination

Lab 8

SPI Light Sensor

These materials produced in association with Imagination.
Join our University community for more resources.
community.imgtec.com/university

MIPSfpga Lab 8: SPI Light Sensor

1. Introduction

In this lab you will review and synthesize a configuration of the MIPSfpga system that contains another peripheral, the Digilent Pmod ALS, the Ambient Light Sensor. As in previous labs, in order to integrate a new peripheral into the MIPSfpga system, you will perform 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 the AHB-Lite bus used in MIPSfpga system.
3. Write software support that allows the application program running on the MIPS microAptiv UP core inside MIPSfpga system to drive the peripheral using the corresponding memory-mapped input/output (I/O) registers.

This lab will help you 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. The SPI bus used to communicate with the sensor is an example of a serial bus, where data bits are sent one at a time, 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 the 7-segment displays. By combining a sensor, a system controller, and an output device (the 7-segment display) you will construct a useful gadget, a light meter.

This lab can be further combined with Lab 10: Interrupts, to demonstrate the interrupt-driven approach to I/O used in many real embedded systems.

2. Light Sensor

Figure 1 shows the light sensor used in this lab, the Digilent PmodALS - Ambient Light Sensor. You can order this sensor for about $10 from this website:

http://store.digilentinc.com/pmod-als-ambient-light-sensor

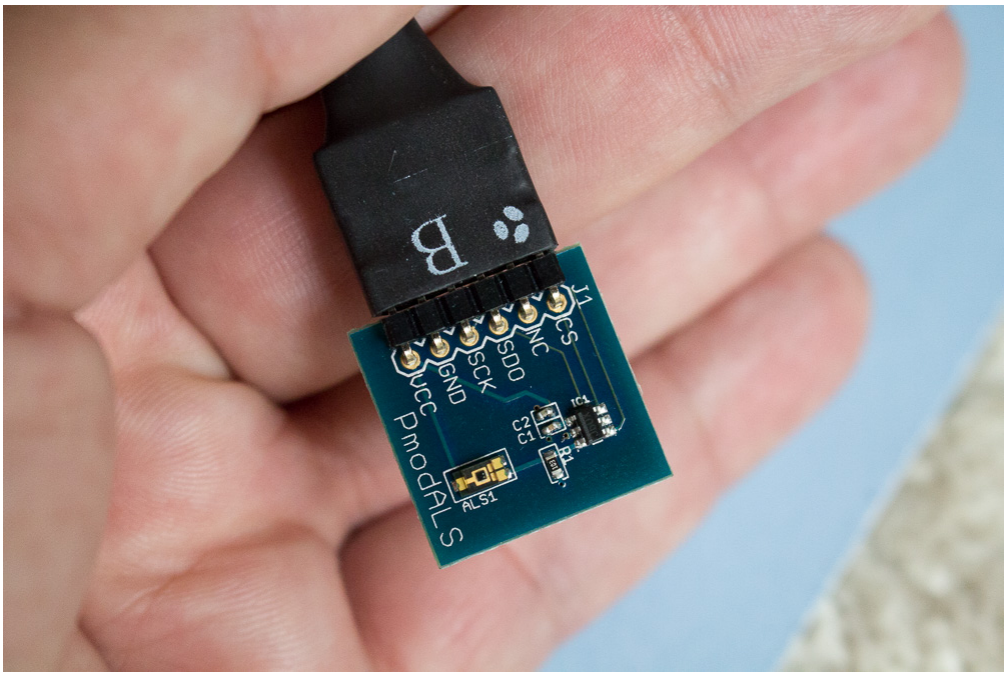

Figure 1. Digilent's PmodALS - Ambient Light Sensor

The light sensor communicates with other devices using a protocol called Serial Peripheral Interface (SPI). This protocol is called serial because it transmits bits sequentially. Serial protocols are convenient for connecting chips on printed circuit boards (PCBs) because they use few pins and, thus, use only a small amount of the limited number of pins available on a typical chip.

SPI is a synchronous serial protocol that is relatively straightforward and fast. It uses only 2-3 pins for the interface: Serial Clock (SCK), Serial Data Out (SDO), and Serial Data In (SDI). One device, called the *master* device, generates SCK and SDO. A second device, called the *slave*, accepts those inputs and sometimes also generates SDI to be fed back to the master device as an input. Figure 2 shows the connection between an SPI master and slave device and the waveforms for the master device's signals. The master communicates 8 bits of data at a time (most significant bit first) to the slave using SCK and SDO. At the same time the slave may send 8 bits of data back to the master on the master's SDI input. SCK asserts a clock edge only when SDO has valid values on it.

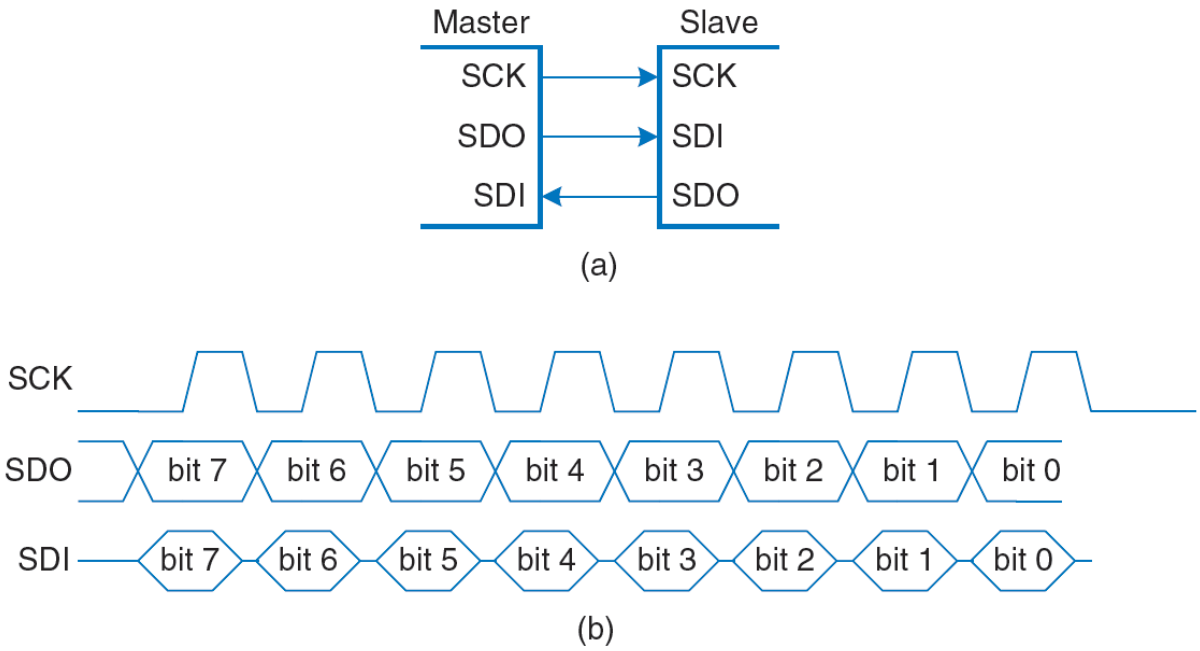

Figure 2. SPI connection and waveforms for master (courtesy *Digital Design and Computer Architecture*, 2nd Edition, Harris & Harris, © 2012 Elsevier)

Additional information about the SPI protocol is available in an article on Digilent's website at https://reference.digilentinc.com/pmod:communication_protocols:spi.

The specific variant of SPI protocol used by the light sensor is described in sensor documentation available here:

https://reference.digilentinc.com/_media/reference/pmod/pmodals/pmodals_rm.pdf

An excerpt from that documentation is shown in Figure 3.

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.

Figure 3. SPI protocol used in Digilent PmodALS - Ambient Light Sensor
(from https://reference.digilentinc.com/_media/reference/pmod/pmodals/pmodals_rm.pdf)

Several other serial protocols are also often used to communicate with sensors, actuators and other computers. Table 1 shows a comparison of the three most popular serial protocols used for simple point-to-point connections in embedded systems: SPI, UART and I^2C.

Table 1. Serial protocol comparison table
(from the 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 various 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 connections inside a typical chip is not critical and maximizing the amount of information transmitted per clock cycle is more important.

In addition, synchronizing signals on multiple parallel wires inside the chip is much easier than outside. Outside the chip, noise and different wire lengths can make synchronization difficult.

For these reasons, on-chip buses tend to be parallel, while off-chip protocols are frequently serial.

As described in the MIPSfpga Getting Started Guide (GSG), the MIPS microAptiv UP core inside the 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 1 shows the general structure of the MIPSfpga system based on AHB-Lite interconnect (from the MIPSfpga GSG_. Details about the protocol are documented in *MIPS32® microAptiv™ UP Processor Core AHB-Lite Interface* manual included into MIPSfpga GSG package.

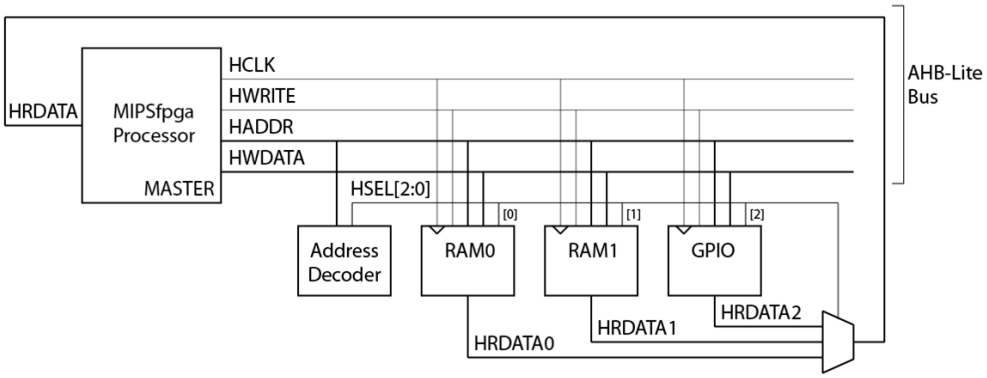

Figure 4. AHB-Lite bus in the MIPSfpga system

Again, as described in the MIPSfpga GSG and in previous labs, 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 shown in Figure 5 for single reads and Figure 6 for single writes.

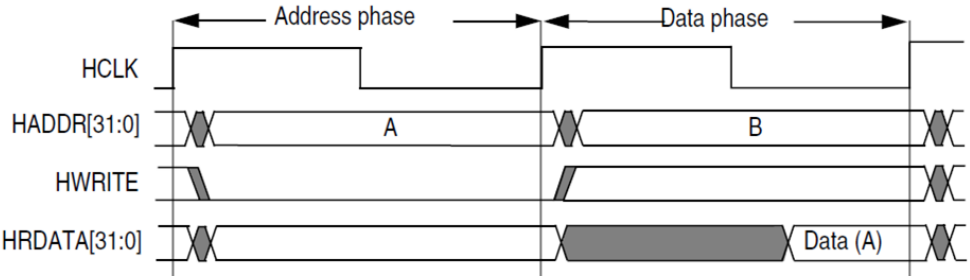

Figure 5. A waveform of a single AHB-Lite read transaction from the AHB-Lite specification

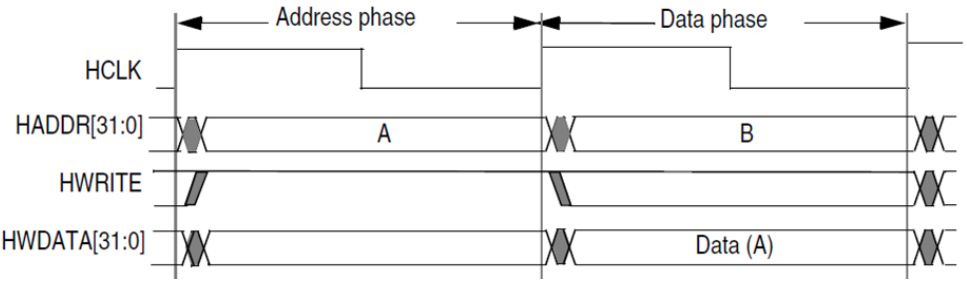

Figure 6. A waveform of a single AHB-Lite write transaction from the AHB-Lite specification

3. Lab Steps

Now that we have discussed the hardware (SPI light sensor) and SPI interface, this section describes how to complete the lab. Almost all generic steps in this lab are the same as in prior labs. The general steps, such as how to synthesize the MIPSfpga system or download a program onto MIPSfpga, are not repeated. Only steps that differ from previous labs are detailed here.

3.1 Software

First review the C program found in the LightSensor\main.c file and shown below. 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: the LEDs and the multiple-digit 7-segment displays.

```
#include "mfp_io.h"

int main ()
{
      // enable the two right-most 7-segment display digits
      MFP_7SEGEN = 0xFC;

    while (1) {
        MFP_LEDS          = MFP_LIGHTSENSOR;
        MFP_7SEGDIGITS    = MFP_LIGHTSENSOR;
    }

    return 0;
}
```

Memory-mapped registers are defined in the header file LightSensor\mfp_io.h (shown below), which is 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 at virtual address 0xBF800014 (physical address 0x1F800014). The C *#define* macro *MFP_LIGHT_SENSOR* makes this register look just like a variable.

```
#ifndef MFP_MEMORY_MAPPED_REGISTERS_H
#define MFP_MEMORY_MAPPED_REGISTERS_H

#define MFP_LEDS_ADDR           0xBF800000
#define MFP_SWITCHES_ADDR       0xBF800004
#define MFP_BUTTONS_ADDR        0xBF800008
#define MFP_7SEGEN_ADDR         0xBF80000c
#define MFP_7SEGDIGITS_ADDR     0xBF800010
#define MFP_LIGHTSENSOR_ADDR    0xBF800014


#define MFP_LEDS                (* (volatile unsigned *) MFP_LEDS_ADDR          )
#define MFP_SWITCHES            (* (volatile unsigned *) MFP_SWITCHES_ADDR      )
#define MFP_BUTTONS             (* (volatile unsigned *) MFP_BUTTONS_ADDR       )
#define MFP_7SEGEN              (* (volatile unsigned *) MFP_7SEGEN_ADDR        )
#define MFP_7SEGDIGITS          (* (volatile unsigned *) MFP_7SEGDIGITS_ADDR    )
#define MFP_LIGHTSENSOR         (* (volatile unsigned *) MFP_LIGHTSENSOR_ADDR   )
```

3.2 Hardware

In this lab we don't need to handle all cases of the SPI protocol. A generic flexible interface module would be quite long and complicated, and 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* (the “chip select”) signal goes low. This specific version of the 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 (found in Lab08_SPILight\VerilogFiles\rtl_up\system). 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?

```
module mfp_ahb_spi_light
(
    input             clk,
    input             resetn,
    output            cs,
    output            sck,
    input             sdi,
    output reg [15:0] value
);

    reg [21:0] cnt;
    reg [15:0] shift;

    always @(posedge clk or negedge resetn)
    begin
        if (~resetn)
            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 clk or negedge resetn)
    begin
        if (~resetn) begin
            shift <= 16'h0000;
            value <= 16'h0000;
        end
        else if (sample_bit) begin
            shift <= (shift << 1) | sdi;
        end
        else if (value_done) begin
            value <= shift;
        end
    end

endmodule
```

3.3 Glue Logic to Interface SPI with AHB-Lite Fabric

Now add glue logic to interface the SPI with the AHB-Lite Fabric. For example, modify the configuration parameters in the file *system/mfp_ahb_const.vh* file to memory-map the SPI light sensor output. You will want to add the following lines:

```
`define H_LIGHTSENSOR_ADDR     (32'h1f800014)
`define H_LIGHTSENSOR_IONUM    (5'h5)
```

Also modify the top-level module, *mfp_nexys4_ddr*, that instantiates a board-independent module *mfp_sys* and connects the following GPIO pins to the SPI I/O, as shown in Table 2. Remember to also modify the constraints file (mfp_nexys4_ddr.xdc).

Table 2. Module and Nexys4 DDR Board Connections

PmodALS Light Sensor Name	MIPSfpga Module Name	Nexys4 DDR Board Pin	Input/Output of Nexys4 DDR
SPI_CS	SPI_CS	JA[1]	Output
SPI_SCK	SPI_SCK	JA[4]	Output
SPI_SDO	SPI_SDI	JA[3]	Input

Now modify mfp_sys.v, mfp_ahb_with_loader.v, mfp_ahb.v, and mfp_ahb_gpio.v to add the new SPI interface module that works with the light sensor.

After you have finished making hardware changes, simulate, debug, and synthesize the updated MIPSfpga system and download it onto the Nexys4 DDR FPGA board. Hook up the PmodALS Light Sensor to the FPGA board as described in Table 2 and shown in Figure 7. Then download and compile the software as you have done in prior labs.

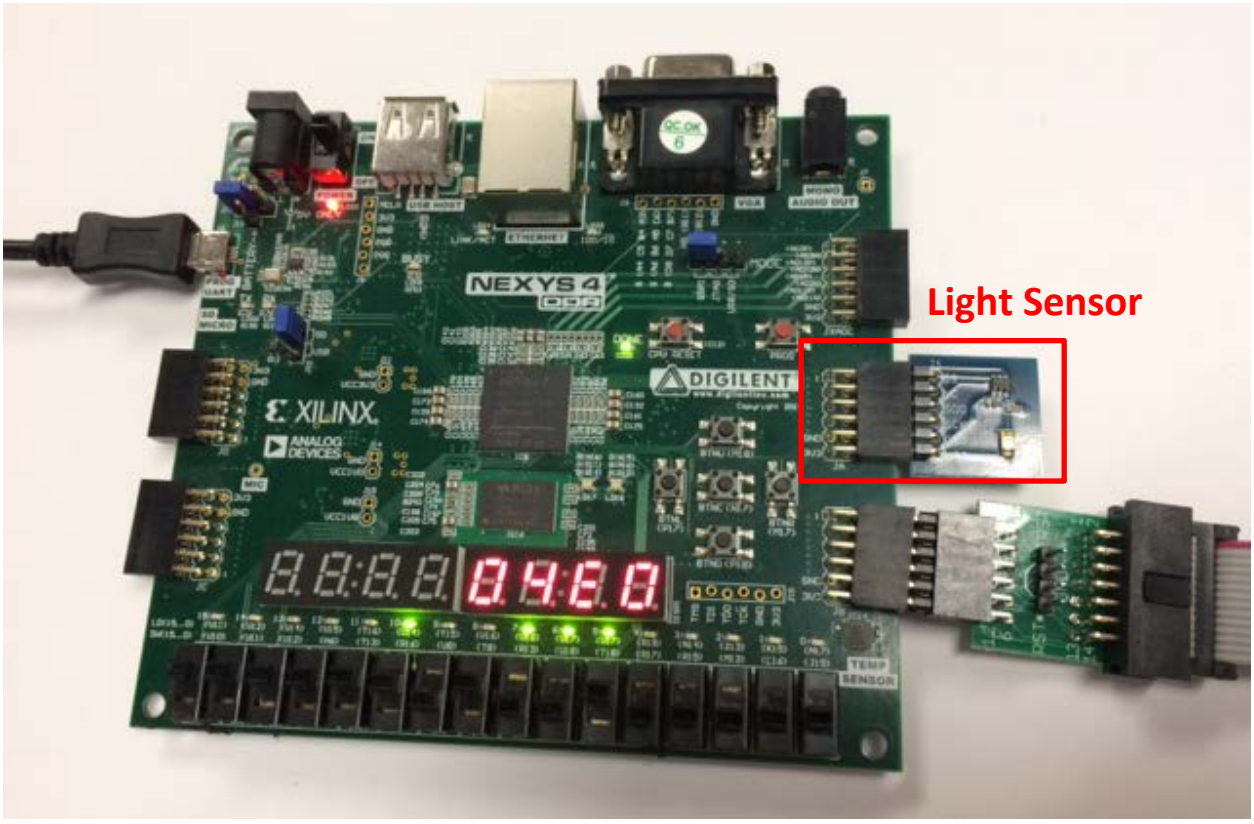

Figure 7. Nexys4 DDR connected to PmodALS Light Sensor (and Bus Blaster probe)

4. Follow-up Exercises and Projects

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

After completing Lab 10, 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 the *SI_Int* signal of the MIPS microAptiv UP core. Measure the system performance improvement that comes from offloading input-output to the interrupt service routine.

You can use this lab and the interrupt lab (Lab 10) as examples to integrate additional sensors and actuators into the MIPSfpga system. Many companies, including Digilent, a National

Instruments company, offer several peripheral modules that can be relatively easily integrated into MIPSfpga, such as the peripherals shown in Figure 8. These modules can be ordered from http://store.digilentinc.com/pmod-modules.

Figure 8. Peripheral modules from Digilent (photo courtesy Digilent Inc.)