****

**Lab 5**

**Memory-Mapped I/O: 7-Segment Displays**



**MIPSfpga Lab 5:**

**Memory-Mapped I/O: 7-Segment Displays**

# Tareas que debes realizar en esta práctica:

1. Lee las partes subrayadas del documento proporcionado a través del Campus Virtual (“Chapter-4-Hardware-description-languages\_2007\_Digital-Design-and-Computer-Architecture\_Edited”), comparando los ejemplos en *VHDL* (que ya debes conocer de anteriores asignaturas) con los de *Verilog* (que es el *HDL* utilizado en el *soft-core* de *MIPSfpga*). Posteriormente puedes volver a acudir a este documento para consultar las dudas que te vayan surgiendo.
2. Lee con detenimiento la Sección 1 (“MIPSfpga Memory-Mapped I/O”) de este documento. Posteriormente puedes volver a acudir a esta sección para consultar las dudas que te vayan surgiendo.
3. Analiza el *SoC- Base* (i.e. el que implementaste en la Práctica 1), que se muestra en la Figure 1, tratando de entender a fondo los siguientes módulos (durante esta práctica consideraremos el *MIPSfpga Processor* como una caja negra):
	1. mfp\_sys (nos interesa solo porque instancia mfp\_ahb\_withloader)
	2. mfp\_ahb\_withloader (nos interesa solo porque instancia mfp\_ahb)
	3. mfp\_ahb
	4. mfp\_ahb\_gpio
4. Analiza el *SoC* con soporte para los displays de 7 segmentos, que se proporciona en el fichero comprimido *Part2\_IO.rar*. Descomprime este fichero y ubícalo dentro del directorio *MIPSfpga\_Labs\Labs\Xilinx*. Trata de entender a fondo los siguientes módulos (ubicados en el directorio *MIPSfpga\_Labs\Labs\Xilinx\Part2\_IO\Lab05\_7seg\HDLFiles\rtl\_up\system*):
	1. mfp\_sys (nos interesa solo porque instancia mfp\_ahb\_withloader)
	2. mfp\_ahb\_withloader (nos interesa solo porque instancia mfp\_ahb)
	3. mfp\_ahb (igual que el anterior; solo cambia el interfaz)
	4. mfp\_ahb\_gpio (ahora el GPIO gestiona, además de los LEDs, botones y switches, los displays de 7 segmentos)
	5. mfp\_ahb\_sevensegtimer (Controlador HW de este nuevo periférico)
	6. mfp\_ahb\_sevensegdec
5. Crea en Vivado el nuevo SoC, siguiendo los pasos del primer guion de la Práctica 1 “MFP\_Lab01\_VivadoProject”, pero utilizando los nuevos módulos proporcionados.
6. Analiza el programa disponible en el directorio *MIPSfpga\_Labs\Labs\Xilinx\Part2\_IO\Lab05\_7seg\CExample* y, una vez que lo hayas entendido bien, ejecútalo en el nuevo SoC.
7. Crea un programa en ensamblador de MIPS que muestre la secuencia 7-6-5-4-3-2-1-0 en los 8 displays de 7 segmentos, mostrando cada dígito en un display (i.e. el display de la izquierda mostrará el 7 y el display de la derecha mostrará el 0). La secuencia se mostrará toda al mismo tiempo y se mantendrá indefinidamente.

# MIPSfpga Memory-Mapped I/O

A processor uses the memory interface to interact with peripheral devices, such as the switches, LEDs, and 7-segment displays on the Nexys4 DDR FPGA board. *Memory-mapped I/O* enables a processor to write to or read from a peripheral device in the same manner that it reads or writes memory. Each peripheral device is assigned one or more memory addresses. When the processor accesses such a memory address, the peripheral device is accessed instead of memory. The MIPSfpga system uses the AHB-Lite bus to access external memory and peripherals.

## AHB-Lite Bus

The AHB-Lite bus has a clock, write enable, address, and read and write data signals (HCLK, HWRITE, HADDR, HRDATA, and HWDATA), as shown in Figure 1. The "H" prefix indicates that they are part of the AHB-Lite bus. Memory and peripherals are connected to this interface to receive and supply data. The MIPSfpga core sends these signals to the AHB-Lite Bus:

* **HCLK**: the 50 MHz system clock
* **HWRITE**: write enable (1 when writing, 0 when reading)
* **HADDR**: the address being read or written
* **HWDATA**: the data being written on a write

The MIPSfpga core receives the following input from the AHB-Lite bus:

* **HRDATA**: the read data produced by memory or the peripherals

The MIPSfpga system has three modules on the AHB-Lite bus: two memories (RAM0 and RAM1) and a general-purpose I/O module (GPIO). RAM0 contains the boot code and RAM1 contains the user code and data. The GPIO unit interfaces with the LEDs, switches, and pushbuttons on the Nexys4 DDR board.



Figure 1. MIPSfpga processor with three peripheral devices

In addition to the three peripherals, the memory-mapped I/O interface requires an Address Decoder and a multiplexer. Depending on the address generated by the processor (HADDR[31:0]), the Address Decoder will enable the processor to access one of the three modules. The Address Decoder generates a select signal HSEL[2:0] that is used by the modules and by the 3:1 multiplexer.

RAM0 is 1 KB and holds the boot code (virtual addresses 0xbfc00000-0xbfc003fc = physical addresses 0x1fc00000-0x1fc003fc). RAM1 is 256 KB and holds the user code (virtual addresses 0x80000000-0x8003fffc = physical addresses 0x00000000-0x0003fffc). The LEDs, switches, and pushbuttons on the Nexys4 DDR board are mapped to virtual memory addresses 0xbf800000-0xbf800008, as shown in Table 1. The processor code uses virtual memory addresses, and the AHB-Lite bus receives physical addresses. The memory management unit (MMU) on the MIPSfpga core performs this address translation.

Table 1. Memory addresses for Nexys4 DDR FPGA board

|  |  |  |  |
| --- | --- | --- | --- |
| **Virtual address** | **Physical address** | **Signal Name** | **Nexys4 DDR** |
| 0xbf80 0000 | 0x1f80 0000 | IO\_LED | LEDs |
| 0xbf80 0004 | 0x1f80 0004 | IO\_SW | switches |
| 0xbf80 0008 | 0x1f80 0008 | IO\_PB | U, D, L, R, C pushbuttons |

The following sequence of MIPS assembly instructions writes the value 5 to the LEDs:

 lui $8, 0xbf80 # $8 = 0xbf800000 (address of LEDs)

 addi $9, $0, 5 # $9 = 5

 sw $9, 0($8)

Recall that load-upper-immediate (lui) loads the 16-bit value 0xbf80 into the upper half of $8 and clears the lower half. Upon execution of the store word instruction (sw), HADDR = 0x1f800000, HWRITE = 1, and HWDATA = 5. The Address Decoder detects that address 0x1f800000 belongs to the general-purpose I/O (GPIO) peripheral and asserts HSEL[2], the select signal associated with that peripheral. The GPIO module detects that HSEL[2] and HWRITE are asserted. Because the GPIO module could potentially write to multiple peripherals, the module uses the address to determine that the LEDs should be written with the value on the HWDATA bus. Specifically, a register whose output is physically connected to the LEDs is updated with the value on HWDATA. That way, the value persists until the LEDs are written again by a later instruction.

Similarly, the following sequence of code reads the value of the switches:

 lui $8, 0xbf80 # $8 = base address of the I/O

 lw $9, 4($8) # $9 = value of the switches

Upon execution of the load word instruction (lw), HADDR = 0x1f800004 and HWRITE = 0 (indicating a read). The Address Decoder detects that address 0x1f800004 belongs to the GPIO peripheral and asserts HSEL[2] (and keeps the other select signals HSEL[1] and HSEL[0] low). The GPIO module detects the address corresponding to the switches and selects to send the value of the switches to its read data output, HRDATA2. The select signals HSEL[2:0] control the multiplexer. Because HSEL[2] is asserted, the multiplexer sends HRDATA2 through to HRDATA. The MIPSfpga processor then reads the value on HRDATA, as it would with a typical read from memory, and stores that value in $9. Thus, after the lw completes, $9 contains the value of the switches.

The hardware for the MIPSfpga AHB-Lite modules is located in the mfp\_ahb module and its submodules. It is best to view this module in your Vivado project, so that the hierarchy is clear. Open the Vivado project that you created in Lab 1. In the Project Manager window, expand mfp\_nexys4\_ddr → mfp\_sys → mfp\_ahb\_withloader → mfp\_ahb to view the mfp\_ahb hierarchy, as shown in Figure 2. Double-click on any of the modules and the Verilog file will open in the neighboring panel.

The mfp\_ahb\_withloader interfaces the AHB-Lite bus with the UART as well to allow direct memory access between the UART and the MIPSfpga memories.



Figure 2. mfp\_ahb hierarchy shown in Vivado

Double-click on mfp\_ahb to view the FPGA board interface signals (see Figure 3). Notice all of the AHB-Lite signals from Figure 1 (HCLK, HADDR, HWRITE, HWDATA, and HRDATA). Additional AHB-Lite signals are also available and will be discussed in later labs. The module also has the memory-mapped I/O signals IO\_Switch, IO\_PB, and IO\_LED that connect to the switches, pushbuttons, and LEDs on the Nexys4 DDR board.



Figure 3. mfp\_ahb interface signals

The modules instantiated within mfp\_ahb are the three peripherals (boot memory, program memory, and GPIO interface), address decoder, and multiplexer shown in Figure 1. The corresponding Verilog module names are given in Table 2. View the Verilog code to see how the functionality described above is implemented.

Table 2. AHB-Lite Modules

|  |  |
| --- | --- |
| **Name from Figure 1** | **Module Name** |
| RAM0 | mfp\_ahb\_b\_ram |
| RAM1 | mfp\_ahb\_p\_ram |
| GPIO | mfp\_ahb\_gpio |
| Address Decoder | ahb\_decoder |
| Multiplexer (for HRDATA) | ahb\_mux |

The GPIO module (mfp\_ahb\_gpio) interfaces with the general-purpose I/O on the Nexys4 DDR board. The MIPSfpga system includes access to the LEDs, switches and pushbuttons on the board. In this lab, we will expand the MIPSfpga functionality to extend to the eight 7-segment displays available on the Nexys4 DDR board.

## 7-Segment Displays

Digits can be represented using 7-segment displays, as shown in Figure 4. Each of the seven segments is labeled a through g. The numbers 0 through F light up the segments shown in Figure 5. For example, the number 0 lights up all but the middle segment, g.



Figure 4. Seven-segment display



Figure 5. Seven-segment display function

Given an input number ranging from 0x0 – 0xF, we will show how to expand the MIPSfpga system to drive the 7-segment displays to show that number. Each segment of the display is low-asserted, so it turns ON when it is 0.

The truth table below (Table 3) shows the inputs (a 4-bit value from 0-15) and outputs for a *7-segment display decoder* (implemented in file **mfp\_ahb\_sevensegdec.v**) that takes in a 4-bit number and produces the value of the segments corresponding to that number. So, for example, with an input of "0", the 7-segment display decoder turns all but the middle segment (Sg) ON. Thus, the first row for Hexadecimal digit "0" shows all the segments as 0 except Sg. (Remember that the segments are low-asserted, so they are ON when they receive 0.) The digit "1" should only have Sb and Sc ON (so Sb and Sc are 0 in that row), and so forth.

Table 3. Truth table for 7-segment display decoder

|  |  |  |  |
| --- | --- | --- | --- |
| **Hexadecimal Digit** | **Inputs** | **Outputs** |  |
| **D3** | **D2** | **D1** | **D0** | **Sa** | **Sb** | **Sc** | **Sd** | **Se** | **Sf** | **Sg** | **HEX** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 01 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 4f |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 12 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 06 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 4c |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 24 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 20 |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0f |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0c |
| A | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 08 |
| B | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 60 |
| C | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 72 |
| D | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 42 |
| E | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 30 |
| F | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 38 |

## 7-Segment Displays on the Nexys4 DDR board

The Nexys4 DDR board has eight 7-segment digits. All eight of the digits on the Nexys4 DDR board are connected to the same low-asserted segment pins, referred to as CA, CB, CC,…,CG. However, each digit has its own enable which is also low-asserted. Figure 6 shows the eight 7-segment displays on the Nexys4 DDR board. CA is connected to the cathode of the A segment for all eight displays, CB to the cathode of the B segment for all displays, and so forth. Each digit has an enable signal, corresponding to the respective bit of the signal AN[7:0]. AN[7:0] is connected via an inverter, to the anode of all segments for the respective digit. For example, if AN[7] is 0, digit 7 will be driven to the values on CA…CG.



Figure 6. Eight 7-segment displays on the Nexys4 DDR board

(© Nexys4 DDR Reference Manual)

To drive each segment to a different value, the enables (AN[7:0]) and segment values (CA…CG) must be driven sequentially, at a rapid enough speed that our eyes don't detect the flicker. For example, to drive display 0 and 1 to the values 3 and 9, we drive CA…CG to display the value 3, and then drive AN[0] LOW, then we drive CA…CG to display the value 9 and drive AN[1] LOW. If we refresh each digit about every 2 ms, our eyes can't detect any flicker.

The module provided in file **mfp\_ahb\_sevensegtimer.v** receives the number to display on each of the eight digits (DIGITS[3:0] to DIGITS[31:28]) and a signal indicating which of the eight displays are enabled (EN[7:0]). It also receives the 50 MHz clock (clk) and a low-asserted reset signal (resetn) as inputs. The outputs are the 7-segment display enables (DISPENOUT[7:0]) and the values of the 7 segments, A-G (DISPOUT[6:0]). Each enabled digit is driven sequentially about every 2 ms.

## Adding Seven-Segment Display Functionality to the GPIO AHB-Lite Module

Once we have the hardware modules that will write the eight 7-segment displays, we must add functionality to the MIPSfpga system to interface with the displays. The goal is to enable the user to write to the eight 7-segment displays using sw:

1. Assign memory-mapped I/O addresses to the enable signal and each of the eight digits
2. Modify the GPIO module to detect these addresses and store the written data to the associated memory-mapped I/O registers
3. Connect these registers to the mfp\_ahb\_sevensegtimer module
4. **Assign memory-mapped I/O addresses**

Assign two addresses to the seven-segment displays: one for the enable and one for the value of the digits, as shown in Table 4. The value to write to the digits is mapped to a 4-bit portion of the DIGITS\_N[31:0] signal. The value of digit 0 is in DIGITS\_N[3:0], digit 1 is in DIGITS\_N[7:4], and so on. The user writes to these addresses to set the enables and the digit values.

Table 4. Memory addresses for Nexys4 DDR FPGA 7-segment displays

|  |  |  |  |
| --- | --- | --- | --- |
| **Virtual address** | **Physical address** | **Signal Name** | **Nexys4 DDR** |
| 0xbf80 000c | 0x1f80 000c | SEGEN\_N[7:0] | AN[7:0] |
| 0xbf80 0010 | 0x1f80 0010 | DIGITS\_N[31:0] | Value of digits 7:0 |

To define these memory-mapped addresses, modify the mfp\_ahb\_const.vh Verilog header file. In Vivado, open Project1. Browse to **mfp\_ahb\_const.vh** in the Project Manager window (as shown in Figure 7), under Design Sources → Verilog Header.



Figure 7. mfp\_ahb\_const.vh Verilog header file

Define the new memory-mapped I/O addresses for the 7-segment displays as H\_7SEGEN\_ADDR and H\_7SEGDIGITS\_ADDR. The Address Decoder (ahb\_decoder module) uses the most significant bits of the address to detect which of the three AHB slaves to enable (the reset RAM, program RAM, or GPIO module). Then, once selected, the GPIO module uses the lower bits of the address to determine which of its peripherals should be written or read. Bits 5:2 of the memory-mapped I/O address are saved in another constant: H\_\*\_IONUM, lower in the mfp\_ahb\_const.vh file, as shown below:

`define H\_LED\_IONUM (4'h0)

`define H\_SW\_IONUM (4'h1)

`define H\_PB\_IONUM (4'h2)

For example, the switches are mapped to physical address 0x1f800004, so bits 5:2 are 0x1 (i.e., H\_SW\_IONUM is 4'h1).

The I/O numbers for the 7-segment display variables are named: H\_7SEGEN\_IONUM and H\_7SEGDIGITS\_IONUM. For example, because the address for the 7-segment digit enables is 0x1f80000c, H\_7SEGEN\_IONUM is 0x3.

**2. Modify the GPIO module**

Then we modify the GPIO module to detect the nine memory-mapped I/O addresses and write the data (HWDATA) to those registers when the corresponding address is detected. In file **mfp\_ahb\_gpio.v,** you can see new signals IO\_7SEGEN\_N[7:0] and IO\_7SEG\_N[6:0]; in a higher-level module, these will drive the enables (AN[7:0]) and segment values (CG…CA) on the Nexys4 DDR board.

Note that there are also two registered signals, one that holds the values of the enables and the other that holds the value of the eight 7-segment display digits. The user will write to these registers using memory-mapped I/O. The registered signal that holds the enables is called SEGEN\_N[7:0]. The register that holds the eight 4-bit values to display on the eight digits is called SEGDIGITS\_N[31:0]. The GPIO module makes that these registers get written when the correct address is detected.

## Connect the 7-Segment Display Signals to the Nexys4 DDR Board

Finally, we must connect the output signals from the AHB GPIO module through to drive the 7-segment displays on the Nexys4 DDR board. To do so, the following modules are modified:

* **mfp\_ahb.v**
* **mfp\_ahb\_withloader.v**
* **mfp\_sys.v**
* **mfp\_nexys4\_ddr.v**
* **testbench.v**

In the highest-level module (mfp\_nexys4\_ddr.v), the output signals that will drive the 7-segment display are named: CA, CB, CC, CD, CE, CF, CG, and AN[7:0]. Recall that CA…CG drive the segments and AN[7:0] drives the enables.

The Xilinx Design Constraint (.xdc) file must also be modified. Double-click on the mfp\_nexys4\_ddr.xdc file to open it. The XDC file assigns the signal names, AN[7:0] and CA – CG, to pins on the Artix-7 FPGA that are physically connected to the 7-segment display inputs on the Nexys4 DDR board using wire traces.

With the .xdc file open in Vivado, search for (ctrl-F) "segment" to find the listing of 7-segment display outputs. For example, signal CA that drives the A segment of the 7-segment displays connects to pin T10 on the FPGA by the following line:

set\_property -dict { **PACKAGE\_PIN T10** IOSTANDARD LVCMOS33 } [get\_ports { **CA** }]; #IO\_L24N\_T3\_A00\_D16\_14 Sch=ca

This pin is connected via a wire trace to the segment A input of the 7-segment displays on the Nexys4 DDR board. CB should output to the R10 Artix-7 pin, and so on. The signals driving the anodes of the 7-segment displays (AN[7:0]) are also assigned Artix-7 package pins.