MIPSfpga by Imagination

Lab 7

Advanced I/O: Audio

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

MIPSfpga Lab 7: Advanced I/O: Audio

1. Introduction

In this lab you will learn the following: (1) how to add another memory-mapped peripheral, a buzzer, to the MIPSfpga system, (2) how to physically connect external peripherals to the MIPSfpga system running on the Nexys4 DDR board, and (3) how to create signals with a specified frequency. In the end, you will write a C program that plays a song.

To complete this lab, you need a buzzer. For example, Table 1 shows an example of one available from Digikey.

Table 1. Buzzer information

Part	Description	Supplier	Price
Buzzer: Part # 102-1153-ND	Magnetic 3.3 V buzzer	Digikey: http://www.digikey.com/product-search/en/audio-products/buzzers/720967?k=102-1153-ND	$1.14

2. Buzzers

A buzzer is a 2-pin device, as shown in below, that produces noise by driving one of its pins at a specified frequency. The other pin is connected to ground. For example, driving one of the buzzer pins at 440 Hz produces an A. The datasheet for the buzzer is in the Lab07_Buzzer folder.

Figure 1. Buzzer from Digikey (Part # 102-1153-ND)

3. Connecting the Buzzer to the MIPSfpga System and Nexys4 DDR Board

To drive the buzzer from a program running on the MIPSfpga system, we need to do three things:

Step 1. Build underlying hardware for driving the buzzer pin at a specified frequency
Step 2. Memory-map the buzzer so that a user can specify the desired frequency
Step 3. Connect the buzzer physically to the Nexys4 DDR board

Step 1. Build underlying hardware for driving the buzzer pin at a specified frequency

First, write the mfp_ahb_buzzer hardware module to generate an oscillating signal to drive the buzzer at a desired frequency. Browse to the Lab07_Buzzer directory and open mfp_ahb_buzzer.v in the VerilogFiles\rtl_up\system subdirectory. You will see the module declaration shown below.

```
module mfp_ahb_buzzer(input        clk,
                      input        resetn,
                      input [31:0] numMicros,
                      output       buzz);
```

The mfp_ahb_buzzer module has a clock and low-asserted reset inputs, `clk` and `resetn`. The clock is running at 50 MHz on the Nexys4 DDR board. It also receives a `numMicros` input that indicates the number of microseconds in *half* of the cycle. The `buzz` output will connect to one pin of the buzzer and should oscillate at the frequency specified by `numMicros`. For example, to drive the buzzer at 440 Hz, the cycle time is 1/440 = 0.002272 seconds = 2,272 µs. Thus to buzz at this frequency, the module should receive 2,272/2 = 1136 on the `numMicros` input. The `buzz` output will then toggle (from 1 to 0, or from 0 to 1) every 1,136 microseconds, creating a period of 2,272 µs, i.e. a frequency of 440 Hz. When `numMicros` is 0, the buzzer shouldn't output a tone.

Fill in the body of the mfp_ahb_buzzer module. Create other submodules as needed.

Step 2. Memory-map the buzzer so that a user can specify the desired frequency

After writing the hardware to support driving the `buzz` output at a specified frequency, memory-map the `numMicros` input to the mfp_ahb_buzzer module to 0xbf800038. You will need to modify mfp_ahb_const.vh, mfp_ahb_gpio.v, and mfp_ahb_withloader.v. See Lab 5 for a review of how to memory-map a peripheral. The user code will write the value of `numMicros` to this memory-mapped address. For example to oscillate the buzzer at 440 Hz, the MIPS assembly code would be:

```
lui   $12, 0xbf80      # $12 = base address of I/O
addiu $13, $0, 1136    # $13 = 1136
sw    $13, 0x38($12)   # write 1136 to numMicros
```

After memory-mapping `numMicros`, test the functionality of the buzzer hardware and memory-mapped interface by simulating some simple MIPS assembly code, for example, you could write an expanded version of the MIPS assembly code above, placed at 0xbfc00000. The code should write different values to `numMicros`. Test that the `buzz` output oscillates at the desired frequencies.

Step 3. Connect the buzzer physically to the Nexys4 DDR board

After the memory-mapped hardware support has been designed and tested above, the last step is to connect the `buzz` output to a pin on the Nexys4 DDR board. Map the `buzz` output to the **PMODC** connector **pin 1** on the Nexys4 DDR board (see Figure 2). The PMODC connector is also referred to as the PMOD JC connector in the Nexys4 DDR manual. Connect the other pin of the buzzer to **ground** (**pin 5** of the **PMODC** connector). First, modify the mfp_sys and mfp_nexys4_ddr modules to route the buzz output all the way out to the Nexys4 DDR FPGA board. You will also need to update the Xilinx Design Constraints (.xdc) file to map the signal name in the wrapper file to the appropriate FPGA pin (i.e., uncomment that needed pin). Then physically connect the buzzer to that pin (and ground).

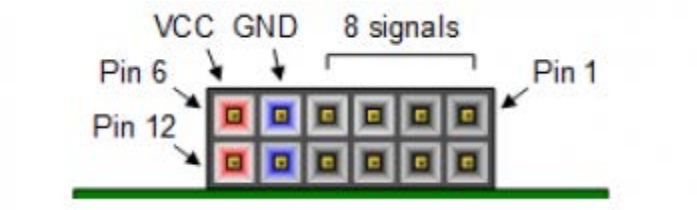

Figure 2. PMOD connector on Nexys4 DDR board (from Digilent's Nexys4 DDR User's Manual)

4. Program for Driving the Buzzer

Now that the hardware support is in place for driving the buzzer, write a program that plays some tones on the buzzer. You will likely want to use the delay_ms function from Lab 6 to set the duration of the note. To do so, write two helper functions: `tone`, and `noTone`. The `tone` function takes the frequency (in Hertz) as input and writes the appropriate value to the `numMicros` memory-mapped register.

```
void tone(unsigned int freq) { }
```

The user plays a tone on the buzzer by calling this function, with the frequency as the argument. For example, the following call to the `tone` function should play the note A (440 Hz) on the buzzer:

```
tone(440);
```

The `noTone` function has no arguments and makes the buzzer silent.

```
void noTone() { }
```


5. Play a song on the MIPSfpga system using the Buzzer

Now write some code to play the following song on your enhanced MIPSfpga system. The notes in the song have the frequency and duration of indicated in Table 2. Put a 10 ms pause between each note. Can you name the song?

Table 2. Notes to play on your MIPSfpga system

Frequency	Duration (ms)
294	220
294	220
392	970
587	470
523	135
494	135
440	135
784	970
587	470
523	135
494	135
440	135
784	970
587	470
523	135
494	135
523	135
440	1470

6. Write Your Own Song

Now write your own song and play it on the MIPSfpga system. You may find the following table of notes and frequencies useful.

Table 3. Notes with corresponding frequencies

Note	Frequency
C	262
C#	*278*
D	294
D#	*311*
E	330
F	349
F#	*370*
G	392

G#	*415*
A	440
A#	*466*
B	494

