MIPSfpga by Imagination

Lab 6

Memory-Mapped I/O: Reaction Timer

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

 © Imagination Technologies 2017

MIPSfpga Lab 6:

Memory-Mapped I/O: Reaction Timer

1. Introduction

In this lab you will add your own memory-mapped I/O peripheral to the MIPSfpga system. In the process, you will learn about counters and how to use functions provided by built-in C libraries.

This lab guides you through the process of adding a new peripheral device: a millisecond counter that counts the number of milliseconds since the processor was reset. You may also wish to refer to the tutorial on adding memory-mapped I/O in Lab 5. At the end of this lab you will write a C program that tests your new memory-mapped counter (as well as your lightning reflexes).

2. Counters

Many digital systems use counters. For example, a computer uses a counter to keep track of the current instruction being executed; a microwave uses a counter to time how long to heat your dinner; and so forth. Most embedded systems give users access to counters for performing timing critical functions.

In this lab you will extend the memory-mapped I/O to include a millisecond counter. This counter returns the number of milliseconds since the MIPSfpga system was reset.

The changes required to the hardware and software to support this millisecond counter are similar to those described in Lab 5 for the 7-segment displays. You will need to modify the following files to support the counter:

- mfp_ahb_const.vh
- mfp_ahb_gpio.v
- mfp_ahb.v

Before modifying these files, copy the MIPSfpga_Labs\rtl_up directory and call it rtl_up_millis.

You will also need to add the mfp_ahb_millisCounter module. Below is the module declaration. The module has two inputs: the on-board 50 MHz clock (`clk`) and the low-asserted system reset (`resetn`). The module should output the number of milliseconds since reset (`millis`).

```
module mfp_ahb_millisCounter(
                    input               clk,
```

 © Imagination Technologies 2017

```
input              resetn,
output reg [31:0]  millis);
```

The millisecond counter is internal to the MIPSfpga system (i.e., its value is read only by the program running on the MIPSfpga core – it is not fed to any external pins), so the Xilinx Constraints File (mfp_nexys4_ddr.xdc) does not need to be modified.

Memory-map the millis output to virtual address 0xbf800034 (physical address 0x1f800034). A read to that address should return the number of milliseconds since reset.

After creating the hardware, follow similar steps as those described in Lab 5 for testing the new hardware. Run a simple MIPS assembly program in simulation (without bootcode) for testing the new hardware. Then write a more complex program for testing the timer that you can load using the Bus Blaster probe.

3. C Program using Counters

After you have finished designing and testing the counter hardware, write a `delay_ms` function with the function declaration shown below.

```
void delay_ms(unsigned int num);
```

The function should use your new milliseconds counter to delay the program for `num` milliseconds.

4. Built-in C Libraries

Now, we discuss the last missing piece in building your reaction timer: C Standard Libraries. In order to test your reaction time, you must be presented with a stimulus at a random time. You will use C's standard libraries to generate a random number and then combine this with your millisecond counter to turn the random number into a random delay.

C provides commonly used functions in *standard libraries*. Standard libraries include the standard I/O library (stdio.h), the math library (math.h), the string library (string.h) and the standard library (stdlib.h). You can google these library names to see a listing of the functions available in each. Appendix C in *Digital Design and Computer Architecture*, Harris & Harris 2012, also provides a description of some commonly used standard library functions.

For example, C provides the `rand` function, in stdlib.h, that produces a random number. We show how to use this function with example code. Browse to this folder:

MIPSfpga_Labs\Labs\Xilinx\Part2_IO\Lab06_ReactionTimer\Random

Open main.c, as shown below.

```
#include <stdlib.h>
#include "mfp_io.h"

void delay();

//------------------
// main()
//------------------
int main() {
  volatile unsigned int val = 0;

  // enable 7-segment display 0
  MFP_7SEGEN = 0xFE;

  while (1) {
   val = rand() % 10;
   MFP_7SEGDIGITS = val;
   delay();
  }
  return 0;
}
```

The compiler directive at the top of the file `#include <stdlib.h>` allows the program to use functions available in C's standard library, in this case the `rand` function. This program writes a random value between 0 and 4 to the 7-segment displays with a delay between each new number. You can replace the delay function with the `delay_ms` function you wrote above if you'd like.

Load and run this program on the MIPSfpga system. Rerun the program several times. In the gdb command shell, type the following command to restart the program:

```
mo reset run
```

As you rerun the program repeatedly, you will notice that the pattern is actually *pseudorandom*: it repeats the same sequence of random numbers each time the program runs.

To make the pattern truly random each time the program runs, you can use the `srand` function, also available in `stdlib.h`. `srand` *seeds* the random number generator with a value. In order to also make this nondeterministic the seed must be random. For example, the program can read the milliseconds timer when a user presses a button.

Browse to the MIPSfpga_Labs\Labs\Xilinx\Part2_IO\Lab06_ReactionTimer\RandomSeed directory and open main.c, as shown below.

```
int main() {
  volatile unsigned int val = 0;
  volatile unsigned int pb = 0;

  // enable 7-segment display 0
  MFP_7SEGEN = 0xFE;

  fdc_init();
  fdc_printf("Waiting for pushbutton press...\n");

  while (pb == 0) {
    pb = MFP_BUTTONS;
    val++;
  }
  srand(val);

  while (1) {
   val = rand() % 10;
   fdc_printf("Random value: %d\n", val);
   MFP_7SEGDIGITS = val;
   delay();
  }
  return 0;
}
```

This program increments a number (`val`) until the user presses a key (i.e., pushbutton). Then it seeds the random number generator with `val`. Re-run this program several times (either by pressing the red CPU RESET button on the Nexys4 DDR board or typing in the gdb window: `mo reset halt` followed by `mo reset run`). Notice that the random sequence is truly random, instead of deterministic as in the previous Random program.

5. Reaction Timer

Now you are ready to write a C program to test your reaction time. Your program should test your (and your friends') reaction time by using the millisecond counter you built and the standard library functions `rand` and `srand`.

To begin the game, the user presses one of the keys (pushbuttons BTNU, BTND, BTNL, BTNR, or BTNC). Then, after a random time between 0 and 3 seconds, turn on the LEDs. The user then presses one of the pushbuttons as fast as possible. The 7-segment displays should display the time it took the user to press the button (in milliseconds) after the LEDs turned on. The user plays the game again by pressing any of the keys.