MIPSfpga
by Imagination

Lab 2

C Programming

MIPSfpga Lab 2: C Programming

1. Introduction

In this lab you will learn to program the MIPSfpga processor in C. You will first complete a tutorial on writing, compiling, and downloading an example C program. Then you will write your own C program to calculate the Fibonacci numbers.

2. MIPSfpga C Tutorial

The MIPSfpga processor is programmed using Imagination's Codescape compiler tools. If you have not already, install the Codescape SDK and OpenOCD by running the installer located in the MIPSfpga Getting Started Guide distribution:

MIPSfpga_GSG\Scripts\Codescape\OpenOCD-0.10.0-img-Installer.exe

If you used a version earlier than 2.0 of the MIPSfpga Getting Started Guide, you will need to update the Codescape/OpenOCD installation by running the installer indicated above.

Codescape supports programming in both C and assembly language. You will use C in this lab and MIPS assembly language in Lab 3.

In this tutorial, you will learn to write and compile a simple program that reads the value of the switches on the Nexys4 DDR board and flashes their values to the LEDs. You'll also learn to step through a program and debug it using the Bus Blaster probe and gdb, which is part of the Codescape tools.

Example C Program

Before writing your own program, we walk you through the steps of compiling, debugging, and running a program using some example code. Browse to this directory:

```
MIPSfpga_Labs\Labs\Xilinx\Part1_Intro\Lab02_C\ReadSwitches
```

Open the file main.c using a text editor such as Notepad or Wordpad. The main.c file contains the ReadSwitches program, as shown in Figure 1.

Compiling, Running, and Debugging

Now compile, run, and debug the ReadSwitches example C program on the MIPSfpga core using the following steps, described in detail below.

Step 1. Download the MIPSfpga system onto the Nexys4 DDR board
Step 2. Compile the C program
Step 3. Load the C program onto MIPSfpga using Bus Blaster
Step 4. Debug the C program using gdb as needed

Remember, that to complete these labs you need to have installed all of the required software and drivers (Vivado, Codescape, and OpenOCD, as well as Bus Blaster and Nexsy4 DDR board drivers) as described in the MIPSfpga Getting Started Guide.

Step 1. Download the MIPSfpga system to the Nexys4 DDR board

First, you will download the MIPSfpga system onto the Nexys4 DDR board. To do so, connect the Nexys4 DDR FPGA board to your computer, turn the board on, and open Vivado. Choose **Flow → Open Hardware Manager** from Vivado's top menu (see Figure 2).

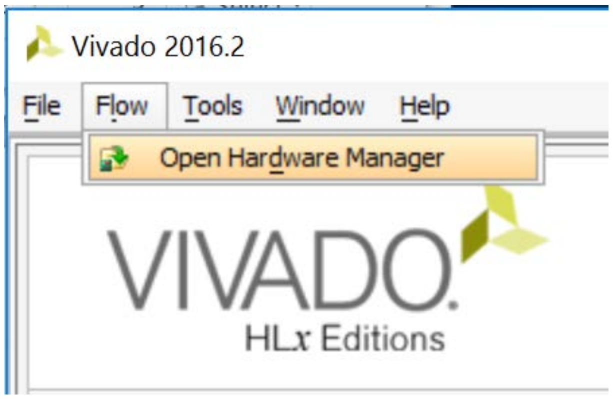


Figure 1. Open Vivado's Hardware Manager

Click on **Open Target → Auto Connect** (see Figure 3).

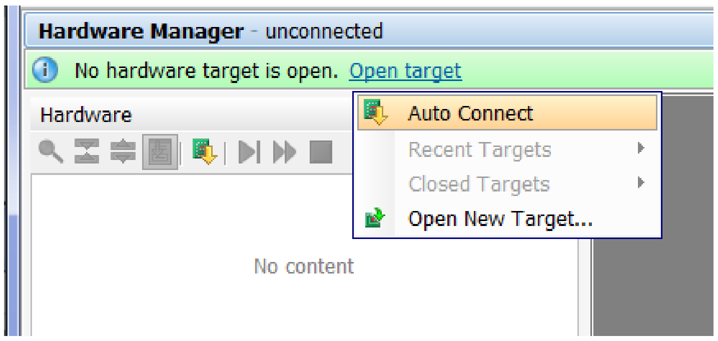
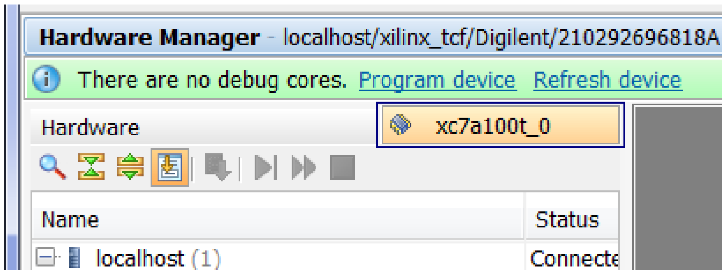


Figure 2. Autoconnect to FPGA on Nexys4 DDR board

Now click on **Program device → xc7a100t_0** (see Figure 4).


Hardware Manager - localhost/xilinx_tcf/Digilent/210292696818A

There are no debug cores. Program device Refresh device

Figure 3. Program device

In the Program Device window, select the bitfile you created in Lab 1 (or the one provided at MIPSfpga_Labs\rtl_up\boards\nexys4_ddr\mfp_nexys4_ddr.bit). Then click Program. Leave the Debug probes file blank.

Click on the red CPU Reset button on the Nexys4 DDR board to reset the MIPSfpga core and begin running the pre-loaded program that displays incremented values on the LEDs.

Step 2. Compile the example program

Now compile the ReadSwitches example C program by opening a command shell. To do so, go to the **Start menu,** type in **cmd.exe**, and select cmd.exe. Or you can shift-right-click on an empty space on your screen and select "Open command window here". In the command shell, change to the Lab02_C\ReadSwitches directory. For example, if MIPSfpga_Labs is in C:\ type:

```
cd C:\MIPSfpga_Labs\Labs\Xilinx\Part1_Intro\Lab02_C\ReadSwitches
```

Compile the example C code by typing make at the prompt in the command window:

```
    make
```

This runs the Makefile, which compiles the user code (found in main.c) with the boot code (found in boot.S). Open and view the Makefile using a text editor such as Notepad, Notepad++, or Wordpad. For future programs, any C program files can be placed under "CSOURCES="

```
    CSOURCES= \
    main.c
```

Below is a brief description of the main parts of the Makefile. The top part of the file gives the names and locations of the compiler tools (gcc, ld, objdump, etc.). These compiler tools are provided with Codescape. They are GNU tools targeted to the MIPSfpga processor.

```
ifndef MIPS_ELF_ROOT
$(error MIPS_ELF_ROOT must be set to point to toolkit
installation root)
endif

CC=mips-mti-elf-gcc
LD=mips-mti-elf-ld
OD=mips-mti-elf-objdump
OC=mips-mti-elf-objcopy
SZ=mips-mti-elf-size
```

The next part of the Makefile indicates the flags to use for compiling and loading the program.

```
CFLAGS = -O1 -g -EL -c -msoft-float -march=m14kc
LDFLAGS = -EL -msoft-float -march=m14kc -Wl,-Map=FPGA_Ram_map.txt
```

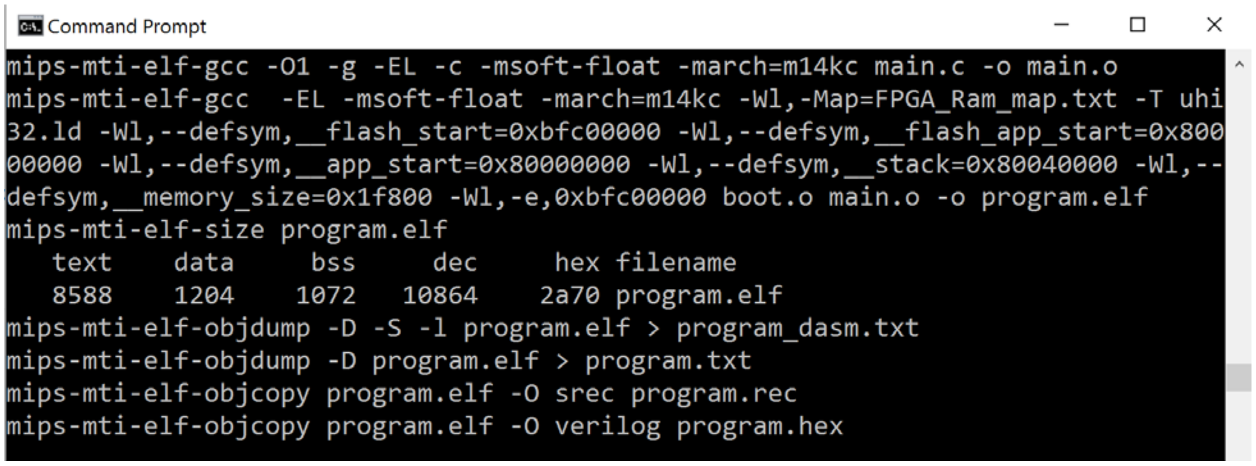
For the C flags, `-O1` says to use optimization level 1. You can change this to higher optimization levels as desired, for example optimization level 2 (`-O2`) or 3 (`-O3`). It is typically a good idea to debug your code using the lowest optimization level (0 or 1). Then, once your code is working, increase the optimization level to produce faster, denser code. `–march=m14kc` indicates to target the M14K MIPS microAptiv architecture, `-msoft-float` says that there is no floating-point unit and to use software floating point routines instead.

The LDFLAGS are used when creating the ELF file, which will provide the program code as well as information about how to load the program and data into the MIPSfpga system's memory. Generally, the LDFLAGS indicate to not generate any floating point instructions (–`msoftfloat`) and to target the M14K MIPS architecture. The file indicated (FPGA_Ram_map.txt) describes how and where the program, boot code, and data will be loaded into memory, according to the physical memory map shown in Lab 1. The rest of the LD flags (`LDFLAGS += ...`) indicate where to place the program, boot code, stack, etc. in memory.

The remainder of the Makefile describes how to compile the program and to clean the directory (i.e., remove files created during compilation). To clean the directory of files created during compilation, type the following at the command prompt:

```
make clean
```

If you just cleaned the directory, **compile the program (main.c) again** by typing `make` at the command prompt. Notice that at the end of compilation, the Makefile outputs the size of the executable, as shown in Figure 5.



```
mips-mti-elf-gcc -O1 -g -EL -c -msoft-float -march=m14kc main.c -o main.o
mips-mti-elf-gcc  -EL -msoft-float -march=m14kc -Wl,-Map=FPGA_Ram_map.txt -T uhi
32.ld -Wl,--defsym,__flash_start=0xbfc00000 -Wl,--defsym,__flash_app_start=0x800
00000 -Wl,--defsym,__app_start=0x80000000 -Wl,--defsym,__stack=0x80040000 -Wl,--
defsym,__memory_size=0x1f800 -Wl,-e,0xbfc00000 boot.o main.o -o program.elf
mips-mti-elf-size program.elf
   text    data     bss     dec     hex filename
   8588    1204    1072   10864    2a70 program.elf
mips-mti-elf-objdump -D -S -l program.elf > program_dasm.txt
mips-mti-elf-objdump -D program.elf > program.txt
mips-mti-elf-objcopy program.elf -O srec program.rec
mips-mti-elf-objcopy program.elf -O verilog program.hex
```

Figure 4. Command shell output of Makefile

The text (instructions) is **8588** bytes, the data is **1204** bytes and the bss segment (static data that should be initialized to 0) is **1072** bytes for a total of **10864** = **0x2a70** bytes. This fits easily within the MIPSfpga memory space. However, you will want to keep your eye on these numbers for larger programs to make sure they fit within MIPSfpga's physical memory.

Note that you can also obtain the size of the program by typing the following at the command line:

```
mips-mti-elf-size program.elf
```

You should now see the following files in the ReadSwitches directory:

- program.elf
- program_dasm.txt
- program.txt
- program.rec
- main.o

program.elf is the main output of compilation. It is the ELF (executable and linkable format) executable that you will use to load the program into the memory of the MIPSfpga core.
program_dasm.txt is a *disassembled* version of the executable. It is basically a human-readable version of the ELF file that shows instruction addresses and instructions interspersed with the line numbers of the higher-level (assembly or C) source code.
program.txt is another human-readable version of the ELF file, but it is not interspersed with the source code information. It shows the memory addresses and corresponding instructions/data, including those memory addresses that should be initialized to 0.
program.rec is the motorola s-record format version of the instructions stored in memory.
main.o is the executable and linkable version of main.c.

Open program_dasm.txt using a text editor to see where the boot code and user code will be placed. The top of the file shows the boot code, starting at 0x9fc00000. Recall that this virtual address maps to physical address 0x1fc000000, which is the physical address of the first instruction fetched upon reset of the MIPSfpga core.

Near the bottom of the file, you can view the user code from main.c, starting at 0x8000075c. This will map to physical addresses starting at 0x0000075c.

Step 3. Load the C program onto the MIPSfpga system using either the Bus Blaster probe or UART

Now that the example program is compiled, load it onto the MIPSfpga core using either the Bus Blaster probe or the UART.

Downloading a program using the Bus Blaster probe

First, connect the Bus Blaster probe to the Nexys4 DDR board, as shown in Figure 6. Do so by connecting the two rows of 6 header pins into the small Adapter Board. Then connect the Adapter Board into the PMOD-B port of the Nexys4 DDR board, as shown in the figure. Connect one side of the ribbon cable into the small Adapter board and the other side into the Bus Blaster probe. Connect one side of the USB cable between the Bus Blaster probe and your computer. Remember, it is best to **use the same port** for the Bus Blaster probe as the one on which you installed the drivers for it.

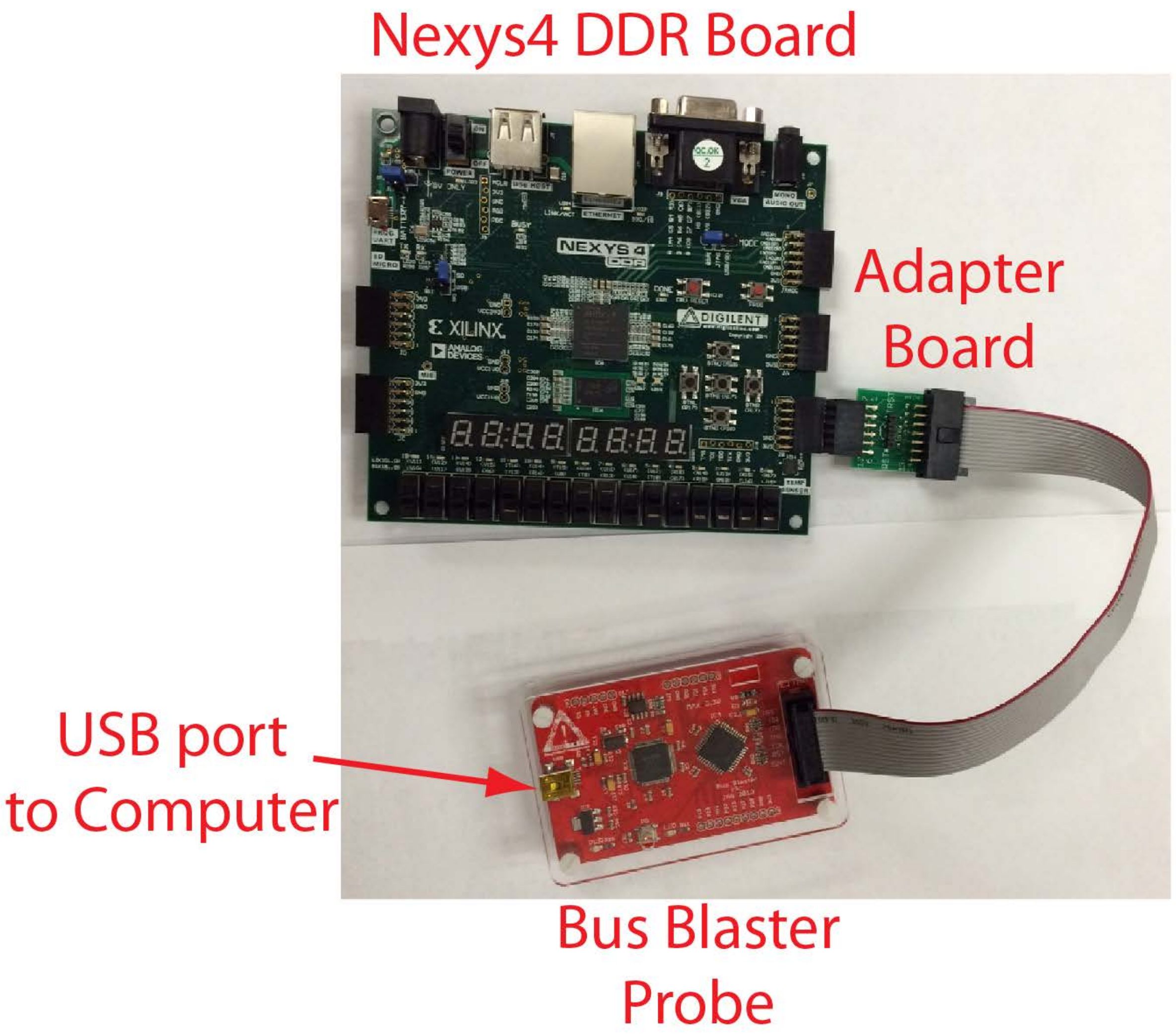


Figure 5. Nexys4 DDR board connected to the Bus Blaster probe

Now open a command shell (i.e., Start menu → cmd.exe.) In the command shell, change to the following MIPSfpga Getting Started Guide directory: **MIPSfpga_GSG\Scripts\Nexys4_DDR** directory. For example, if MIPSfpga_GSG is located on the C drive, type the following at the shell prompt:

```
cd C:\MIPSfpga_GSG\Scripts\Nexys4_DDR
```

Run the **loadMIPSfpga.bat** script that will (1) compile the program, (2) set up a connection to the MIPSfpga core using OpenOCD, (3) download the program onto the MIPSfpga system, and (4) allow you to use the Gnu debugger (gdb) to load and debug the program on the MIPSfpga core. Run the loadMIPSfpga.bat batch script supplied in the directory of the ReadSwitches program as the argument. For example, if MIPSfpga_Labs is on the C drive, at the command font, type:

```
  loadMIPSfpga.bat
C:\MIPSfpga_Labs\Labs\Xilinx\Part1_Intro\Lab02_C\ReadSwitches
```

After running the script, you will see the ReadSwitches program running on the MIPSfpga core. Recall that the ReadSwitches program repeatedly reads the value of the switches and flashes that value on the LEDs. Toggle some of the switches at the bottom of the Nexys4 DDR board and watch as the corresponding LEDs flash.

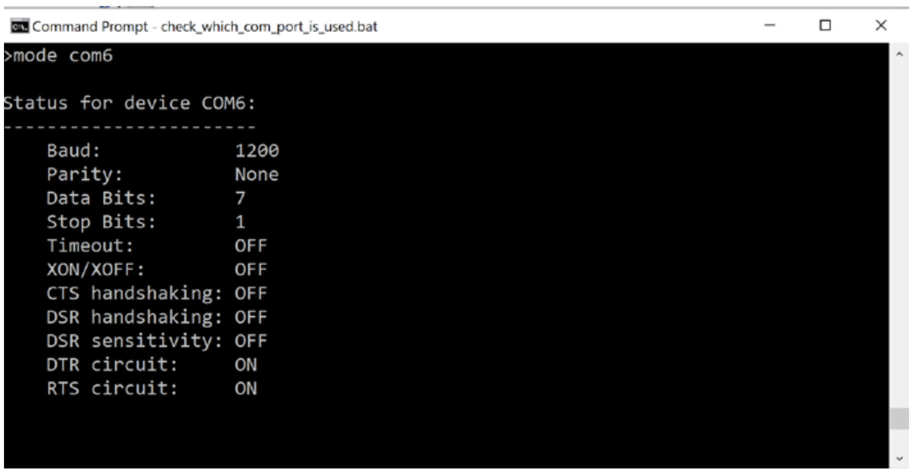
The loadMIPSfpga.bat script first opens a shell to compile the specified program using make. It then opens up two more shells to create the OpenOCD connection and run gdb. After you are finished with a program, close these two windows.

Downloading a program using the UART

The Nexys4 DDR board can use the existing programming cable to download programs. However, you will not be able to use gdb for debugging using this method. Make sure the programming cable is connected between the Nexys4 DDR board and your computer. Now, check which COM port is used by the programming cable or the FTDI cable by running the following batch file at the command prompt:

```
      check_which_com_port_is_used.bat
```

The batch file is available in the MIPSfpga_GSG\Scripts directory and can be run in that same directory. The COM port used by the cable will report as shown in Figure 7. Other ports will report that they are not in use or that it is an illegal device name. In this case, the cable is connected to COM port 6.



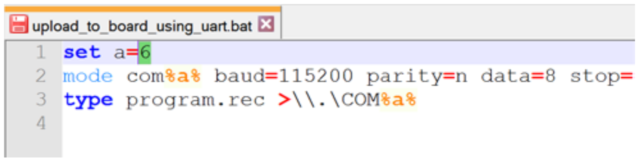
```
>mode com6

Status for device COM6:
----------------------
    Baud:            1200
    Parity:          None
    Data Bits:       7
    Stop Bits:       1
    Timeout:         OFF
    XON/XOFF:        OFF
    CTS handshaking: OFF
    DSR handshaking: OFF
    DSR sensitivity: OFF
    DTR circuit:     ON
    RTS circuit:     ON
```

Figure 6. Checking which COM port is used by the programming/FTDI cable

As another option, you can always use the Device Manager to check which COM port is connected to the Nexys4 DDR FPGA board.

Now download the compiled program to the MIPSfpga system. Make sure that the the program is compiled (i.e., perform `make` in the program directory). Next, edit the **upload_to_board_using_uart.bat** script (available in the MIPSfpga_GSG\Scripts directory) using an editor such as Notepad++. Change the `a` variable to the port used by the cable, in this example 6 (as shown in Figure 8).



```
set a=6
mode com%a% baud=115200 parity=n data=8 stop=
type program.rec >\\.\COM%a%
```

Figure 7. Changing COM port number in upload_to_board_using_uart.bat file

Now copy this file (upload_to_board_using_uart.bat) to the program directory, open a command shell and execute the file. For example, type the following two commands shell:

```
     > cd
C:\MIPSfpga_Labs\Labs\Xilinx\Part1_Intro\Lab02_C\ReadSwitches
     > upload_to_board_using_uart.bat
```

You should now see the program running on the MIPSfpga system on the Nexys4 DDR board.

Step 4. Debug the C program using gdb as needed

Although this ReadSwitches program works and requires no debugging, we will show you how to go through the process of debugging using gdb, supplied as part of the Codescape SDK.

Note that this section can only be completed if you used the Bus Blaster (not the UART) to download the ReadSwitches program onto MIPSfpga.

Click on the gdb command shell that was opened by the loadMIPSfpga.bat script in the previous step. Enter the sequence of commands shown in Table 1 to halt the program, set breakpoints, view variable and register values, etc.

Table 1. gdb command sequence

Command	Description
`Ctrl-c`	Stop the processor by pressing Ctrl-C with the gdb command shell (labeled mips-mti-elf-gdb).
`monitor reset halt`	Reset the processor to the beginning of the program (i.e., pc = instruction address 0xbfc00000).

	Shortcut: `mo reset halt`
`b main`	Set a breakpoint at the main function. (Short for: "break main".) Notice that the breakpoint is set at 0x8000025c. Open the ReadSwitches\program.txt file and search for “main” Note that you can set breakpoints even when the processor is running, but the breakpoints will take effect only when the processor is halted (`mo reset halt`).
`b *0x80000260`	Set a breakpoint at instruction address 0x80000260. In the ReadSwitches C program, this is the load word instruction (`lw`) that reads the value of the switches (see ReadSwitches\program.txt) `   80000260:     8e020004   lw    v0,4(s0)` Note that you could also have typed: `b 19` This would set a breakpoint at line 19 of main.c
`b *0x8000026c`	Set a breakpoint at instruction address 0x8000026c. In the ReadSwitches C program, this is the store word instruction (`sw`) that writes to the LEDs (see program.txt) `  8000026c:      ae020000   sw    v0,0(s0)`
`i b`	List the breakpoints. (Short for: "info breakpoint".) At this point it will list the breakpoint at instruction addresses 0x800007b4 (main), 0x800007b8, and 0x800007c4.
`c`	Continue the processor execution. (Short for: "continue".) It will stop at the first breakpoint, in this case, when it gets to `main` (instruction address 0x8000025c).

`x/3i $pc`	Prints 3 instructions starting with the current instruction ($pc is the program counter and contains the address of the current instruction).
	`8000025c:  3c10bf80   lui   s0,0xbf80`
	`80000260:  8e020004   lw    v0,4(s0)`
	`80000264:  afa20010   sw    v0,16(sp)`

`x/3x $pc`	Prints 3 instructions in hexadecimal, starting at the address specified.
`c`	Continue to the next break point, which is at 0x80000260.
`stepi`	Executes a single instruction. For example, now you will see the PC increment to 0x80000264. Shortcut: `si`
`si`	Step one more instruction. (You can also simply press the Enter key to repeat the last gdb command.)
`p switches`	Now that the switches have been read, we can print the value of the variable `switches`. (Short for: "`print switches`".) For example, if the 3 least significant switches are 1 (i.e., in the UP position),

	`switches` will have the value 7.
`p/x switches`	Prints the value of the `switches` variable in hexadecimal.
`p/x &switches`	Prints the address of the `switches` variable
`i r`	Print the value of all registers. (Short for: "`info registers`".)
`i r v0`	Print the value of register `v0` only. At this point, `v0` holds the value of the FPGA board switches. This value will be written to the LEDs by the store word (`sw`) instruction at 0x8000026c.
`c`	Continue program execution. (Short for: "continue".) Execution is now at 0x8000026c, the store word instruction that will write the value of the switches to the LEDs.
`i r s0`	Print the value of register `s0`. `s0` currently holds the memory-mapped I/O address of the LEDs: 0xbf800000.
`i r v0`	Print the value of register `v0`. `v0` holds the value of the switches that will get written to the LEDs.
`si`	Execute the store word instruction and watch as the LEDs are updated to the value of the switches.
`d 1`	Delete breakpoint 1 (type `i b` to list the breakpoints with their numbers). This deletes the breakpoint at the beginning of main.
`monitor reset run`	Reset and run the processor. This will run the processor without breakpoints, even if breakpoints have been set. **Shortcut:** `mo reset run`

For a list of other gdb commands, refer to the GDB User Manual available as a link on this webpage:

```
http://www.gnu.org/software/gdb/documentation/
```

3. Fibonacci Numbers

Now you will write your own C program, compile it, and run it on MIPSfpga. Create a program that will calculate and display the first 11 Fibonacci numbers on the LEDs. Each number in the Fibonacci series is the sum of the previous two numbers. Table 2 lists the first few numbers in the series.

Table 2: Fibonacci Series

n	1	2	3	4	5	6	7	8	9	10	11	…
fib(n)	1	1	2	3	5	8	13	21	34	55	89	…

fib(n)	0x1	0x1	0x2	0x3	0x5	0x8	0xd	0x15	0x22	0x37	0x59

We can also define the fib function for negative values of n. To be consistent with the definition of the Fibonacci series, what would the following values be?

fib(0) = ____

fib(-1) = ____

These values are useful when writing a loop to compute fib(n) for all non-negative values of n.

Make a copy of the Lab02_C/ReadSwitches folder and rename the new folder Fibonacci. Write your program in the main.c file in that folder. Your program should compute the Fibonacci numbers for n = 1…11 and output each Fibonacci number to the LEDs. The LEDs should display the Fibonacci numbers in binary, with a delay between each number so that they are viewable.

Once you have finished writing your program, use the `make` command to compile it. If there are any errors, fix them and recompile.

After your Fibonacci program compiles without errors, load it onto the MIPSfpga core using either the Bus Blaster probe or the UART interface.

Table 2 lists the Fibonacci numbers in hexadecimal to help you read the binary values on the LEDs. In later labs, you will expand the MIPSfpga hardware to enable you to use the 7-segment displays available on the Nexys4 DDR FPGA board. But for now, remember that if you are using the Bus Blaster probe you can also use breakpoints in gdb to examine the values produced by your program.