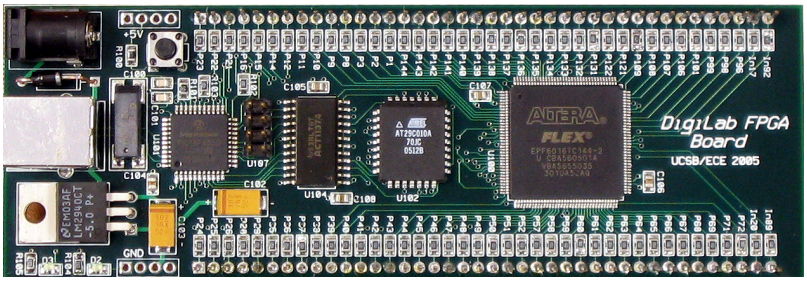
UCSB/ECE DigiLab FPGA Board

Introduction

This site documents the UCSB/ECE DigiLab FPGA Board design, including the software access program 'FPGAtool' and various user-accessible test procedures. The DigiLab FPGA Board has been designed to support the prototyping needs of the Digital Systems Laboratory at UCSB for courses such as ECE 152A (Logic Design) and ECE 152B (Digital Design Methodology).



Snap-shot of board (component-side)

The board features an Altera EPF6016 FPGA, part of the so-called Flex-6000 family of programmable logic chips. This FPGA contains 1,320 logic elements and represents the equivalent of approximately 16,000 logic gates. Packaged in a 144-pin thin-quad flatpack, the FPGA offers 117 user-configurable I/Os. Of these, 80 are brought out to the header strips running along the sides of the board. Also on-board is a 128 Kbyte flash memory that holds the user configuration file for the FPGA so that upon power-on (or manual reset) the FPGA gets configured automatically.

The board is controlled by a small microcontroller (PIC18F4550). This processor was chosen in part because of its built-in USB interface. It operates at 48 MHz and offers non-volatile storage for programs up to 16K instructions in length, a 2 Kbyte static RAM, and 24 highly-configurable I/O ports. The microcontroller contains code to configure the FPGA, to interact with lab PCs over USB (via the 'FPGAtool' program), and to perform extensive testing of the board.

Directions for Use

Board Connections

The DigiLab FPGA board offers 80 user-configurable input/output pins. These pins are located along the top and bottom edges of the board (as seen in the component-side image of the board shown near the beginning of this page). There are three types of pins: column-connected, row-connected, and input-only. The row- and column-connected pins are highly similar, differing only in speed and internal

Downloads

FPGAtool (Ver 1.02/GL) Windows Application
exe | **doc**

FPGA Test: Self-counter
slfctr.rbf

Contacts

Steven Butner

Nitin Kataria

distribution within the FPGA. Any of these pins can be programmed to be an input, an output, or a run-time configurable (tri-statable) I/O pin. As their name implies, there are several **input-only pins** that can only act as inputs. There are four such pins (In17, In20, In89, and In92), two in each row, located on the extreme right-hand end of the pin rows of the card. These input-only pins offer low-skew buffered distribution within the FPGA. They are intended to be used for clock signals, system-wide resets, and any other widely-distributed global signals. The name of each pin (containing its corresponding Altera pin number) is painted on the board near its attachment point.

On the left end of the two pin rows are arrays of power and ground pins. These can be used to supply up to 1 amp of 5V direct current to your DigiLab project breadboard. Overloading these power supply connections will not damage the board but will definitely result in delivery of a voltage lower than 5V on the power output pins. It is highly recommended that you use the FPGA board's 5V DC power and GND connections as your primary power source since such a strategy provides the best possible signal integrity for signals that interact with the FPGA. If for any reason you choose not to use the FPGA board's power supply with your project then you should leave it completely disconnected, **i.e. use it exclusively or abandon it completely.** Do not try to "augment" the FPGA board's supply by attaching another power supply in parallel. Such an arrangement can only cause harm to one of the two power supplies. Since the FPGA is always powered from the on-board supply, **if you use a separate supply for your project chips, be sure that you attach your external power supply's ground terminal to the FPGA board's ground.**

Prototype Board Installation

The Digilab FPGA board has been designed so as to be able to be mounted on a standard student prototyping board, spanning about 10 pin rows (including one of the power/GND distribution rows). Because the FPGA board has nearly 90 pins(!) insertion into and extraction from the prototyping board requires considerable force. Please be extra careful when installing or removing the FPGA board so that neither it nor the prototyping board is damaged. Also note that the pins on the FPGA board are quite sharp, making it easy to cut or scrape yourself during such activities. **Be careful and go slowly!**

Pin Loading

The 80 pins located along the top and bottom edges of the FPGA board can be used to support your digital experiments. The Altera Flex6000 FPGA is a 5V device with standard TTL I/Os. That means an unloaded logic high is approximately 3.75V and a logic low is about 0.1V. The FPGA board has series resistors to protect the on-board circuits from inadvertent damage. These resistors are 1K for most of the user pins, thus limiting the output current to about 4mA per

pin. This should be enough to drive a small number of digital inputs. If a larger fanout is required it may be necessary to use more than one output pin per signal.

There are 10 special higher-drive pins on the board. These are the first ten pins along the top edge --- pins P23, P22, P21, P16, P15, P14, P12, P11, P10, and P9. These special pins have 100 ohm series resistors, thus providing approximately ten times as much drive as regular pins. You should use these pins to drive LEDs or higher-fanout loads. Future versions of the FPGA board will likely have a larger number of high-drive pins, but the first version FPGA board has just 10 such pins.

Detached Mode

The FPGA board provides a reasonably large programmable logic structure that can be used for building digital projects. Normal (detached) operation requires only the connection of a 9-12V DC power source. Once powered, the FPGA board's microcontroller will initialize the FPGA using the user's logic configuration file. This file must have been previously downloaded to the FPGA board using attached mode (see below). Since the file is stored in the board's non-volatile flash memory, it is not necessary to re-download the file until you need to make a change. Even without any connections to a power source, your board will remember the FPGA's configuration file as well as other important properties.

Though it should not be necessary to reinitialize the FPGA as long as the power remains present, the entire board is reinitialized whenever the reset button is pushed and released (or when power to the board is removed and subsequently reapplied).

Attached Mode

The DigiLab FPGA board can be attached to any Windows-XP computer via a USB cable. In this mode the program 'FPGAtool' is required. This program can be downloaded from this website. When 'FPGAtool' is active, it establishes communication with the FPGA board and displays the board's properties (serial number, date/time, filename of its last programming, and other vital statistics). If a previously-attached board becomes disconnected, 'FPGAtool' announces that fact and awaits user intervention. In order to do business in "attached" mode, the board's USB connection must remain intact.

While in attached mode one can download a new FPGA configuration file to the board. During such transactions, LED "D3" blinks. Upon successful completion of a download, LED "D3" should be off. If errors occur during downloading, a message should appear in the FPGAtool message window and LED "D3" will likely remain on. In normal operation, only the power LED ("D2") will be illuminated.

Programming your Design

Overview

At the highest level, the procedure for creating a customized FPGA for your application involves the following steps:

- creating a project under the Altera FPGA design environment, Quartus-II
- writing one or more module descriptions in Verilog or VHDL and associating them with the Quartus-II project
- compiling/synthesizing the design
- assigning FPGA pins to ports of the module(s)
- fitting, assembly, and generation of the programming file
- simulating the behavior of the custom module(s) using ModelSim
- downloading your design to the Digilab FPGA Board

Many of the steps above are discussed in Altera's excellent Quartus-II tutorial.

Assigning FPGA Pins to Module Ports

The Flex6000 FPGA used on the Digilab FPGA Board is housed in a 144-pin package. There are 117 user-programmable pins. Of those, 80 are available at the board interconnect boundary. The pins that are available are those with labels of the form "`P<number>`" or "`In<number>`" located along the header strips on the edges of the board. Any pin whose number does not appear in such a label is not usuable (because there is no user-accessible attachment to it). The complete list of pins that are available for you to use is here.

Once you have chosen which pins to use, the actual assignment process is performed using the Quartus-II Assignment Tool. Pull down the "Assignments" menu and select the "Pins" entry. This should bring up the Assignment Editor. You should be able to find a list of symbols coresponding to the ports of your design. For each port you must assign a physical pin. Be sure the choose pins whose pin numbers appear as labels near the FPGA Board's header strips. **Not all pins have attachments to the headers!** When you have assigned all of your ports to physical pins you can close the Assignment Editor program and move on to the fitting process.

Device Options

The Digilab FPGA Board contains a Flex-6000 FPGA. The specific part number that you must use as a target device is `EPF6016TC144-2`. By clicking on the **Assignment** menu under **Device...** you will gain access to the **Settings** dialog. Make sure that you select the correct device family (FLEX6000) and part number, including the packaging options, as shown below.

Quartus-II Assignment->Device... Settings dialog

Immediately below the **Family** field within the **Settings** dialog is a button labelled **Device & Pin Options**. Select this to gain access to the rest of the device-specific options. Note that these must be selected in exactly the manner described here in order to get correct operation of your design. The **Device & Pin Options** pane should appear as below. The **General** tab has been selected.

Quartus-II Device & Pin Options ... with General tab selected

The on-line program FPGAtool allows you to take advantage of the JTAG boundary scan capabilities of the Flex-6000 FPGA. This can be useful during checkout of your connections on the lab bench. In order to use JTAG boundary scan you must select "**Enable JTAG-BST Support**" as shown above. If left unselected your resulting programming file will still work but you will not be able to utilize any of the JTAG capabilities.

We shall discuss the remaining tabs within this dialog without showing screen shots. Go to each of the tabs discussed and choose the option(s) as specified in order to get correct operation. Under the **Configuration** tab, you must select **Passive Serial** for the Configuration Scheme. Be sure to leave the **Use configuration device** option unselected.

Under the **Programming Files** tab, you must select **Raw Binary File (.rbf)** for the file format. Leave all other programming file format options unselected.

Under the **Unused Pins** tab, we recommend you select the option "**As input tristated**" for the behavior of all unused pins. It is OK to go ahead and familiarize yourself with some of the other choices but the remaining tabs of this dialog either control options that are not applicable to the Flex-6000 device or the settings you may choose should not matter in the application. At this point we should be ready to build the project.

Processing your Design

The Quartus-II software integrates all of the steps needed to compile, fit, assemble, and generate programming files for your design. In general, when you make a change to any part of your design Quartus-II will apply the needed processing the next time you request it to build the design. In order to request such a build, go to the **Processing** menu and simply click on the **Start Processing** button. During processing, progress bars will appear to track completion and status of the various tasks. At the end of processing you will be notified about any error(s) that may have occurred.

The RBF File

Once processing has updated and you have achieved an error-free build, Quartus-II should have produced a programming file in the Raw Binary Format --- a so-called "RBF" file. This file format was chosen in the previous setup steps. You will need the RBF file to load your design onto the FPGA. Use FPGAtool to do this.

Simulating your Design

Before loading your design into the FPGA it is always a good idea to verify the functionality of your synthesized logic through simulation. A "test bench" will be required in order to simulate your module(s). The test bench is a special Verilog (or VHDL) module that instantiates the module being tested, drives its inputs, and observes its simulated outputs.

Actually there are two levels of simluation that one should perform before moving into hardware. The first simulation is known as RTL (register transfer level) simulation. This is done before synthesis to verify that your design has the proper behavior. RTL simulation as well as the second form -- gate-

level or post-synthesis simulation --- can be performed using the same test bench, if desired. Gate-level simulation is performed after synthesis and place/route has been performed. It attempts to model the actual paths and structures that will be used within the FPGA. It is capable of modeling the actual delays that should be expected in your synthesized design.

Creating a Verilog Test Bench

First and foremost a test bench is a regular Verilog file. It does have a few special properties, however.

- It does not have any ports
- It need not be synthesizable Verilog
- It must instantiate the module being tested
- Ports of the module under test are regular signals within the test bench.
- It should include `$stop` at some point in order to terminate the simulation.

Matching the formal port names of the module under test with the signals of the test bench should always be done using the dot notation "`.formal (actual)`". Attempting to match formal ports with actual test bench signals using port definition ordering is discouraged and more often than not simply does not work.

Assume that our design to be tested is a D flip-flop with module name "`dff`" and with formal input ports "`clk`", "`D`", and "`reset`" and with output port "`Q`". The proper way for a test bench to instantiate the `dff` module and attach to its input and output ports would be as shown below.

```
`timescale 1 ns / 10 ps
module testbench;
  reg clock, rst, data;          // stimulii
  wire dffq;                     // test module output label
  dff UUT ( .clk(clock), .D(data), .reset(rst), .Q(dffq) );
       :
endmodule
```

Note that the inputs to UUT (the unit under test) are registered signals of the test bench and that the output of UUT is a `wire`. Because we used the dot-notation, the ports of the `dff` module can be specified in any order. It is usually a good idea to include a `` `timescale `` directive in your test bench since Verilog uses scalable time units. The `timescale` shown above establishes the basic unit as one nanosecond with a fundamental internal resolution of 10 picoseconds. This means that if the test bench specifies a delay of 15 time units using the notation `#15`, it will cause the simulation to delay for 15 nanoseconds.

Adding Stimulii and Controls within a Verilog Test Bench

With the basic test bench structure in place we next turn our attention to the generation of input stimulii. Most digital designs contain a reset signal. In this case the D flip-flop we are simulating is supposed to have an asynchronous, active-low reset signal. Let's begin our test by resetting the D flip-flop.

We begin by using an `initial` block. Such a construct is executed once, at the beginning of simulation. Multiple `initial` blocks are allowed in a test bench. When present, the multiple blocks execute concurrently.

We will begin the test by initializing the signal values within the test bench to appropriate initial values. We then use `#100` to delay 100 ns before the next change, releasing reset. The `initial` block to do this is shown below.

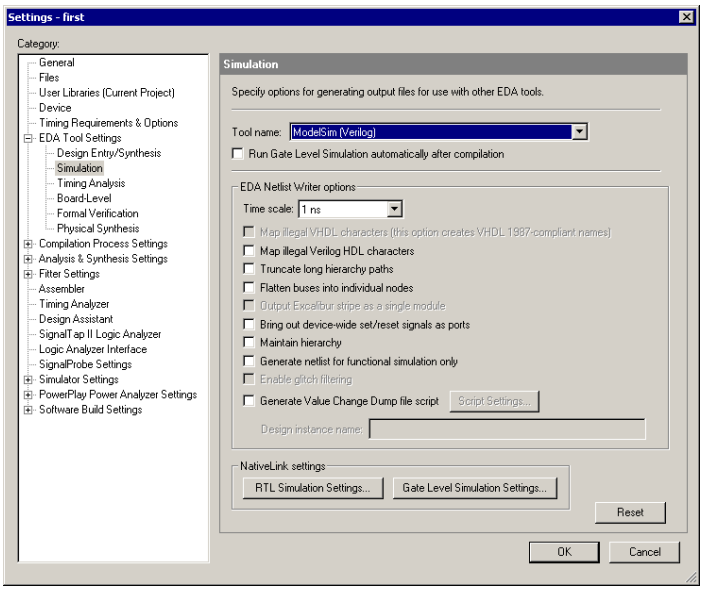
```
initial
begin
  clock <= 1'b0;                // initialize clock low
  reset <= 1'b0;                // initialize reset low (it is active-low)
  data  <= 1'b1;                // initialize D input high
  #100 reset <= '1b1;           // bring reset high after 100 ns delay
  forever #50 clock <= ~clock;  // generate a 10 MHz clock
end

initial #20000 $stop;           // ensure that our simulation ends
```

So far, we've reset everything (with a 100 ns wide active low reset pulse) and started our clock running with a simulated frequency of 10 MHz --- that's done with the `forever` statement. If our test bench had ended there, it would have continued to simulate forever! By adding one additional `initial` block that contains just a single delayed `$stop` statement, we ensure that the simulation will come to an orderly stop at a particular time; in this case it will stop after 20 microseconds. The remaining test stimulii can now be added in one or more additional `initial` blocks, using appropriate `#` delays to place the various test cases as needed. Note that all of the `initial` blocks begin at the same time and execute concurrently. Thus their individual notions of time begin at the same epoch.

Setting up ModelSim

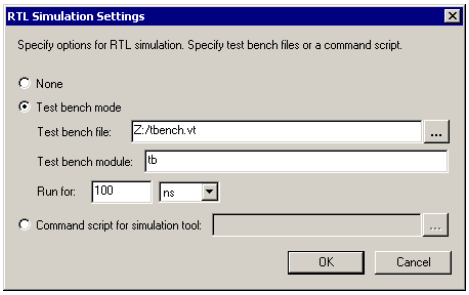
In order to describe to Quartus-II how to launch and run ModelSim/SE with your design you must setup the NativeLink interface. This must only be done once (per Quartus-II project). To setup for either of the two kinds of simulation, go to the Assignments menu and select EDA Tool Settings. The **Settings** dialog box should appear. In the **Category** list,

select **Simulation**. The RTL simulation settings box should appear as below.



Quartus-II Simulation Settings dialog

Near the bottom of the Settings dialog screen are two buttons for setting up the NativeLink interface to ModelSim/SE. One is labeled **RTL Simulation Settings** and the other is **Gate Level Simulation Settings...**. They both lead to similar screens. The RTL Simluation Settings screen is shown below.



Quartus-II RTL Simulation Settings screen

In this screen we establish that we wish to run in test bench mode, i.e. with a Verilog test bench serving as the stimulus for the tests. We specify a file name containing the Verilog test bench (ending with the extension .vt) a module name for the given test bench module, and a time duration for the simulation run.

Once the details have been described to the Nativelink interface one can launch a testbench-based simulation by going to the **Tools** menu of Quartus-II and choosing **EDA Simulation ...**. You will then have the a choice of RTL or gate-level simulation. Once the type of simulation is chosen, ModelSim/SE will be launched and your project and testbench should be attached correctly.