FPGA Analyser & Signal Viewer

Team Mac and Cheese

Senior Design 2024

Tufts University

Professor Steven Bell

ES4 - Introduction to Digital Logic

Table of Contents

Introduction & Motivation

As Tufts undergrads beginning a major in Electrical or Computer Engineering, ES4 is one of the pivotal classes necessary for understanding what will be happening down the line. As an introduction to digital logic the course covers HDL (Hardware Description Language) specifically VHDL (Very-high-speed Integrated Circuit Hardware Description Language). As a description language, the work flow is unlike traditional coding, lacking the visibility of debug statements like printing to the terminal. Instead, VHDL describes a circuit that is then implemented using logic blocks which creates a black box effect that can be difficult to debug.

As a solution to this, we have created a VHDL module with a GUI companion that samples the signals within the circuit and writes them to the user's computer. This is done by sendings data back up through the cable used to flash the program onto the FPGA. The GUI reads this data and displays it, ideally allowing the user to compare the desired result with the actual result.

Source

All files are uploaded and documented in the following Github repository: https://github.com/Polidori-112/MacNCheese

Video Example

The following video walkthrough will explain how to use the module and GUI
Mac and Cheese Logic Analyzer Implementation Tutorial

System Requirements

The VHDL module works on Updiuno 3.0/3.1 using a ICE40UP5K FPGA chip. Any cable capable of connecting to the board and flashing the VHDL will also be capable of communicating with the GUI. The GUI requires a computer running windows with 1GB of space. The GUI runs as an executable, but the python code can be taken from GitHub. Linux is currently not supported.

This manual will focus on Lattice Radiant as the VHDL development platform and programmer. This module should work with any IDE capable of running VHDL 2008.

How to Use - Gui

Download the executable from the Github Releases or ask a TA for access to the executable. When ready to sample, begin the executable and determine which of the two modes you will be using:

> **Live Stream** will give a continuous feed of the program, but is limited by the speed of the transfer rate between the FPGA and the computer. This mode is best used on slower programs (sub 10MHz) or programs that aim to run longer than a minute. The live stream will be saved after the program has been stopped so there is no need to worry about missing signals as they happen.
>
> **Write to Memory** will take a set number of samples, either limited by the cap provided by the user or the size of the onboard RAM. This RAM has a faster write capability then Live Stream so this is ideal for fast programs, but due to the 1Mbit size limitation it can not run for a long time. Aim for a snapshot of quickly alternating signals, such as a VGA frame. The memory will then be read into the GUI and display the waveform in its entirety.

If Live Stream is desired, set the mode bit to 0 and propagate the samples with the bits you would like to be sent to the GUI. Similarly for Write to Memory set the mode bit to 1 and progate with the bits you would like sampled. Set the number of samples you would like. If the number exceeds the available space on the chip, the module will limit the number of samples as needed.

How to Use - VHDL Module

Copy every VHDL code file from the Github for the VHDL module into the project. Write your VHDL as your normally would and then define and instantiate the “sampler” component, it should look like this:

```
—
entity MNC is
generic (
    use_ram      : integer := 0;
    use_ext_clk  : integer := 0;

    byterate     : integer := 100;

    NUM_INPUTS   : natural := 8;
    NUM_SAMPLES  : natural := 1024;
    ADDR_WIDTH   : natural := 10
);
port (
    Data_in      : in std_logic_vector(7 downto 0);
    clk_48       : in std_logic;
    ext_clk      : in std_logic;

    serial_txd   : out std_logic;
    serial_rxd   : in  std_logic;
    spi_cs       : out std_logic
—
```

These variables are as follows:

Instantiation Guide		
GENERICS		
Variable	**Acceptable Values**	**Usage**
`use_ram`	0 or 1	Selects Mode 0 : live stream 1 : RAM
`use_ext_clk`	0 or 1	Selects clock rate 0 : uses byterate 1 : uses external clock (clock mapped in PORTS)
`byterate`	0 - 10000	Sets the byterate if not using external clock
`NUM_INPUTS`	8	Number of input bits (Do not change)
`NUM_SAMPLES`	0 - 125000	Sets number of desired samples (RAM only)
`ADDR_WIDTH`	$\log_2$(NUM_SAMPLES) (rounded up)	Sets RAM address width (RAM only)
PORTS		
Variable	**Signals to connect to**	**Usage**
`Data_in`	Logic vector of bits that you would like to sample	These are the waveforms you will see. You can fill with 0s for unused bits.
`clk_48`	HSOSC output that drives a 48MHz clock	Sets clock rate of the sampler, Upduino can only support one HSOSC so must be externally sourced

`ext_clk`	Any clock signal less than 20kHz	If `use_ext_clk=1` then this will be the sampling rate
`serial_txd`	Pin 14	UART transmit pin
`serial_rxd`	Pin 15	UART receive pin
`spi_cs`	Pin 16 and HIGH	Sets communication Mode

Alternatively you can create a project with the sampler pre-installed by logging into the Halligan servers and running:

```
wget  https://raw.githubusercontent.com/Polidori-112/MacNCheese/main/setup.py
```

Then running:

```
python3 setup.py
```

Then open the project in Radiant and you should see a modified top file and several VHD files needed to run the sampler.

Common Errors

Unable to connect to the UART

- Make sure that the COM port is properly seeing the USB device. The program should automatically understand the data sent but if necessary change port settings to the desired baud rate.
- Make sure the `serial_txd, serial_rxd, spi_cs` are port mapped and `spi_cs` is high

Unable to connect to GUI

- Make sure environment has python3
- Make sure the proper executable has been installed

VHDL/Radiant Issues

- Make sure the top file reflects the necessary pins: `serial_txd, serial_rxd, spi_cs`
    - Ensure said pins are mapped to 14, 15, and 16 respectively
- Make sure the sampler has been defined as both an entity and a component
- If defining generics from external variables, make sure that ‘integers’ and ‘naturals’ are not mixed