

The FPGA

Author: Oussama Sekkat
Graduate student advisor: Thomas Schmid
Advisor: Dr. Mani Srivastava

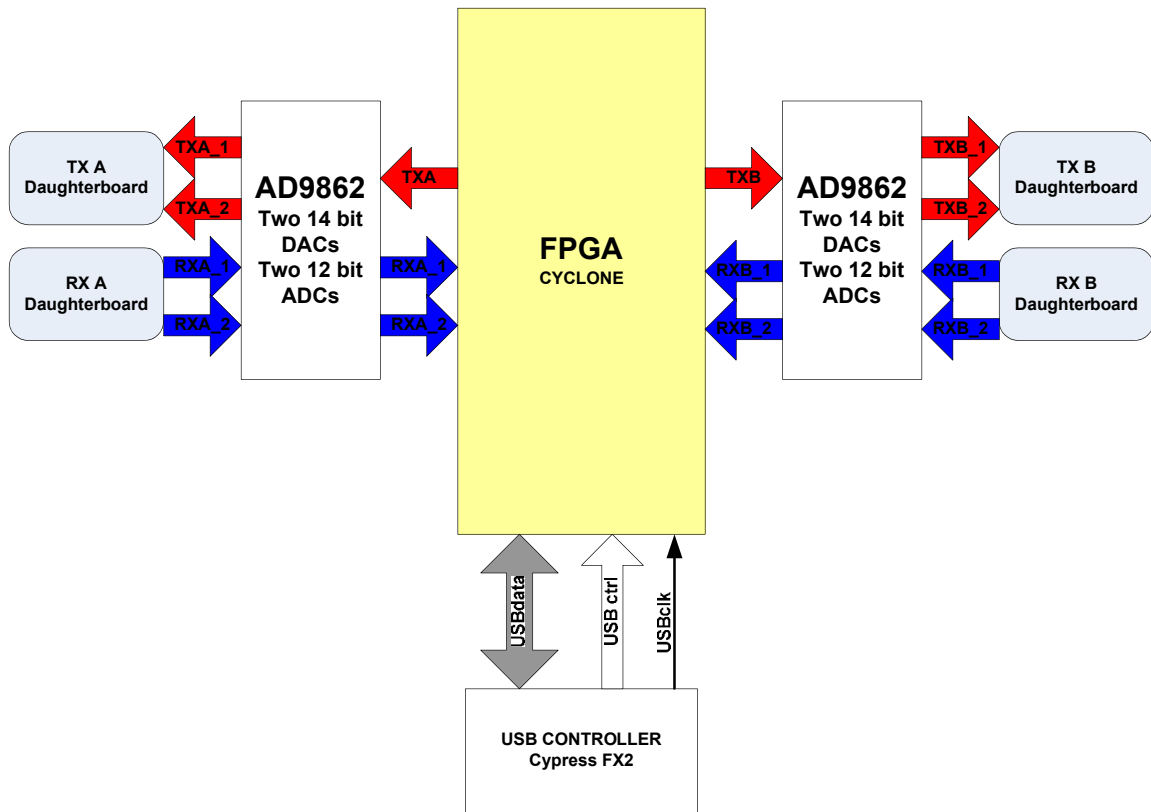
Abstract

This article gives a description of the main functionality of the FPGA in the Universal Software Radio Peripheral (USRP). It also includes a brief tutorial on how to reprogram the FPGA.

The Universal Software Radio Peripheral (USRP)

The Universal Software Radio Peripheral was designed as a low cost board solely for the purpose of running GNU radio applications. Fully developed by Matt Ettus, it is a very flexible platform and can be used to implement real time applications. It is the bridge between the software world and the RF world. The motherboard has a USB controller, an FPGA and two Analog Device (AD9862) chips. The USB controller contains the firmware that defines its behavior and the USB endpoints. The firmware also takes care of loading the FPGA bit stream. The FPGA handles the high bandwidth computations and reduces the data rate to something we can send over the USB 2.0. The Analog Device chip is a mixed signal processor that takes care of the conversion between analog and digital signals, digital up conversion in the transmit path and interpolation/decimation of the signals. The motherboard can have up to 4 daughterboards, two for receive and two for transmit. They consist of the RF front end where the signal is up converted from the intermediate frequency to the carrier frequency or vice versa for the received signal. The USRP and the daughterboards can be purchased from www.ettus.com . That website also contains data sheets for the board.

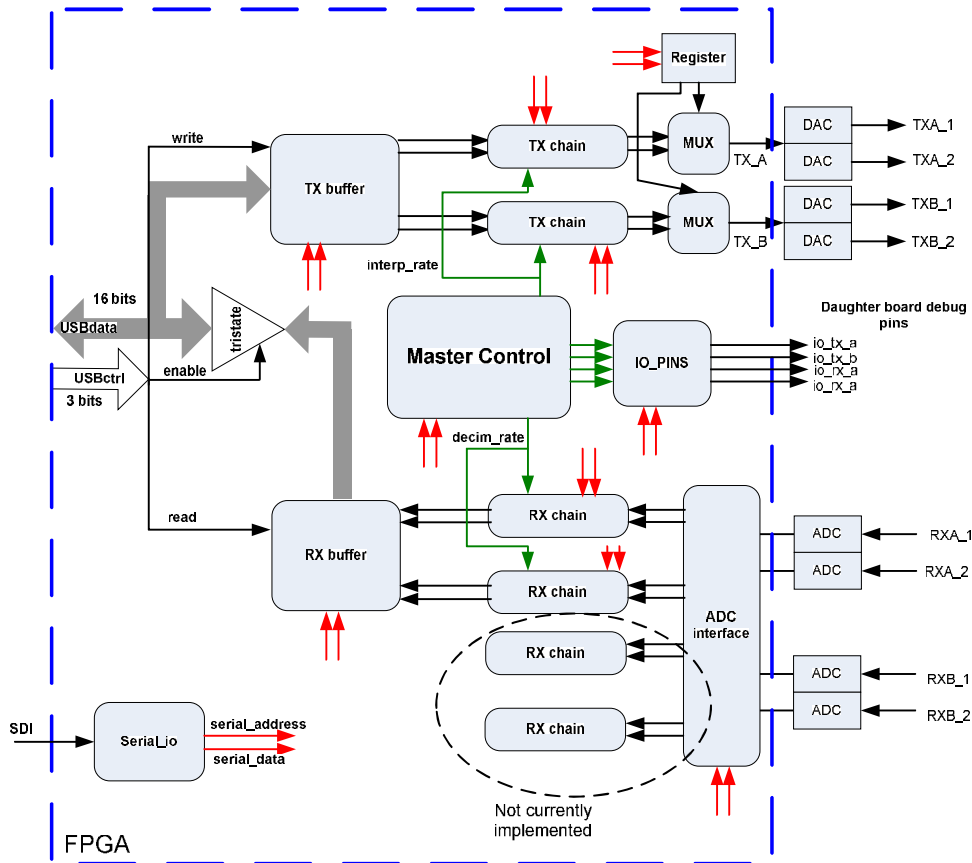
The following is a high level view of the USRP.



Now let's have a look inside the FPGA in order to understand the functionality of each of its building blocks.

Inside the FPGA

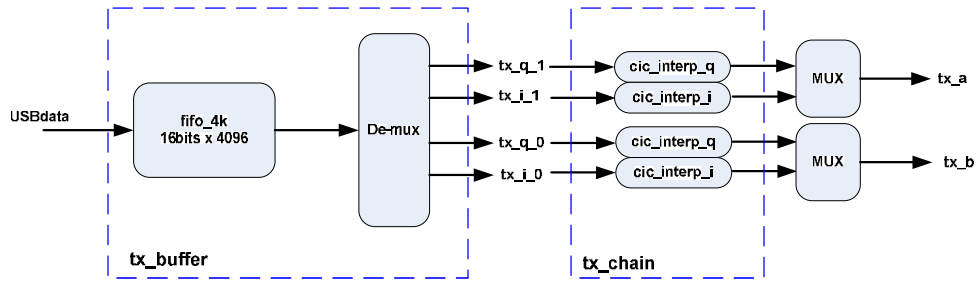
The following figure shows a high level view of the main building blocks inside the FPGA.



The transmit path:

On the transmit side, the data comes as a 16 bit value from the USB. It goes through the TX buffer module. This module demuxes the data to be transmitted and decouples it into I and Q signals. Then, each complex pair of signals I, Q, goes through the TX chain module which interpolates the data to 32 MS/s. Those I, Q signals are then interleaved and sent to the AD 9862 chip where the signals are interpolated by a factor of 4, then up converted to an intermediate frequency and finally converted to an analog signal.

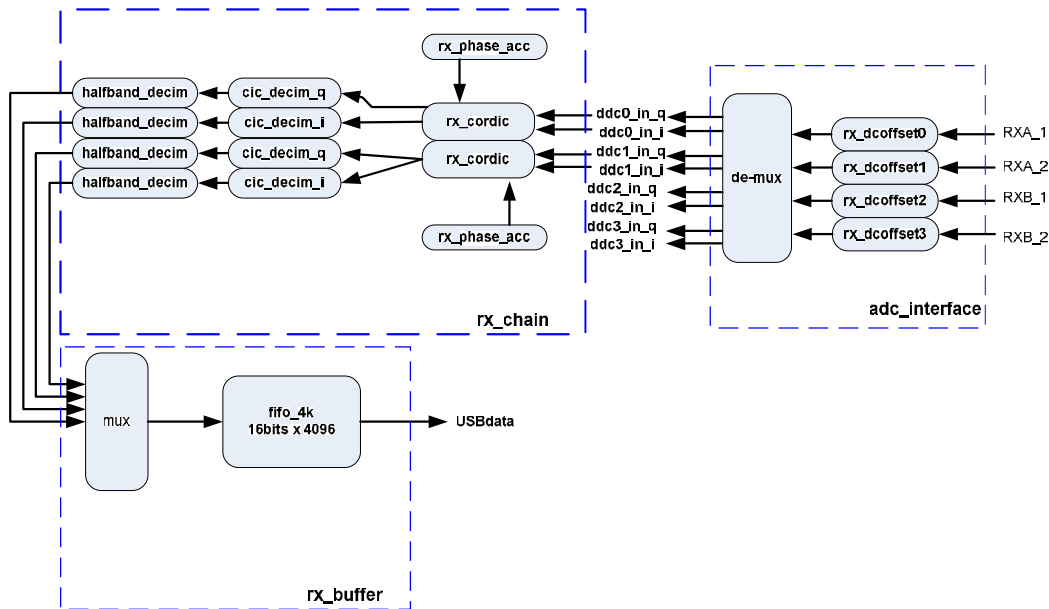
Here is a high level schematic of the transmit path:



The receive path:

The signals from the daughter boards are first converted to a 12 bit digital value in the AD 9862 chip, and decimated in the same chip. The signals then enter the ADC interface module which routes them to the proper digital down converter. Then the RX chain module in the FPGA takes care of the digital down conversion to baseband and decimation to 32MS/s. And finally, the signals go through the RX buffer module where they get interleaved into a 16 bit value. That value is sent to the PC through the USB bus.

Here is a high level schematic of the receive path:



How to compile the verilog code and program the FPGA

The program used to compile the verilog code is Altera Quartus II for windows. A Linux version also exists but is not free. Here is a link to Quartus[®] II Web Edition Software v6.0 Service Pack 1:

https://www.altera.com/support/software/download/altera_design/quartus_we/dnl-quartus_we.jsp

For every Windows machine you run the software on, you will need a license. This license is given for free and you can get it by clicking on the link below the download link.

Once you've installed the software, you can edit verilog code and reprogram the FPGA in the USRP. Now let's go over how to do that.

Editing the verilog code:

First you should copy all the verilog files to your windows machine.


For example, create a folder C:\USRP.

- From your Linux machine, copy all the files in the gr-build\usrp\fpga folder into the new c:\usrp folder.
- There are other verilog files in the gr-build\usrp firmware\include folder. Copy that folder into your c:\usrp folder.
- Now some verilog files will need to be edited because the "include" folder is in a different location and so all the verilog files which have the following statement at the beginning :

``include "../../../../firmware/include/[name].v"` will need to be changed to

``include "c:\usrp\include\[name].v"`

We will come back to that later on.

- You are now ready to open the project. To do that, go to File -> Open Project. Open the project file C:\usrp\toplevel\usrp_std\usrp_std.qpf.
- If you are using the new version of Quartus II 6.0 (as of June 20th 2006) it will mention that the last time that file was opened was with an older version of Quartus. Just ignore that, and compile the code.
- To compile the code go to Processing -> Start compilation or just click on the  icon.

- The compilation will give you lots of errors. That's normal. (I get 46 errors)
- If you click on the error tab you'll see that lots of errors are of the kind: can't open verilog design file "file_name.v". This is what I mentioned previously. To fix that, make sure all such files have the correct path name in their ``include` statement.

For example, the usrp_std.v file has the following statements in the beginning:

``include "usrp_std.vh"`

``include "../../../../firmware/include/fpga_regs_common.v"`

``include "../../../../firmware/include/fpga_regs_standard.v"`

Make sure you change those statements to:

- ```

`include "usrp_std.vh"
`include "C:\usrp\include\fpga_regs_common.v"
`include "C:\usrp\include\fpga_regs_standard.v"

```
- Proceed the same way for all the files that have those kind of include statements and recompile the code.
  - The compile will take anywhere between 5 to 20 min depending on the speed of your processor. At the end, you'll get the compilation report that looks like the following:

| Flow Summary            |                                              |
|-------------------------|----------------------------------------------|
| Flow Status             | Successful - Mon Sep 25 15:56:31 2006        |
| Quartus II Version      | 6.0 Build 202 06/20/2006 SP 1 SJ Web Edition |
| Revision Name           | usrp_std                                     |
| Top-level Entity Name   | usrp_std                                     |
| Family                  | Cyclone                                      |
| Device                  | EP1C12Q240C8                                 |
| Timing Models           | Final                                        |
| Met timing requirements | Yes                                          |
| Total logic elements    | 11,045 / 12,060 ( 92 % )                     |
| Total pins              | 173 / 173 ( 100 % )                          |
| Total virtual pins      | 0                                            |
| Total memory bits       | 139,264 / 239,616 ( 58 % )                   |
| Total PLLs              | 0 / 2 ( 0 % )                                |

- Now, you can program the board using the new bit file.

#### Programming the board:

- Now, you need to transfer the bit file to your Linux machine. I suggest you put your file in the /usr/local/share/usrp/rev{2,4} folder.
- The USRP has almost no non-volatile storage. Thus, whenever you run an application, the FPGA is programmed. By default, the USRP loads the FPGA bitstream from /usr/local/share/usrp/rev{2,4}/\*.rbf. However, if you want the board to load your own bitstream, you can specify it as an additional keyword constructor argument when opening the USRP:  

```
u = usrp.sink_c(0, 64, fpga_filename="usrp_std.rbf")
```

This will load the usrp\_std.rbf bitstream from the /usr/local/share/usrp/rev{2,4} folder.

## Table of contents

- The Universal Software Radio Peripheral
- Inside the FPGA
  - the transmit path
  - the receive path
- FPGA/USB interface (not written yet)
- How to compile the verilog code and program the FPGA