

ratload

Installation and Use

Jacob Hladky
California Polytechnic State University
San Luis Obispo, CA
jhladky@calpoly.edu

February 18, 2015

1	Introduction	2
1.1	Requirements	2
1.2	Project Manifest	2
1.3	Bug Reporting	3
1.4	Contact	3
2	Installation	4
2.1	Integration on RAT CPU	4
2.1.1	Integration of UART	4
2.1.2	UART Verification	6
2.1.3	Integration of new prog_rom module	7
2.2	Installation on GNU/Linux or OS X	7
2.3	Installation on Windows	8
2.4	Serial Configuration	8
2.4.1	Nexys 2	8
2.4.2	Nexys 3	8
3	Use	9
3.1	Using winRATLoad (Windows)	9
3.2	Using ratload (GNU/Linux or OS X)	9

1 Introduction

The ratload system consists of two separate components: a computer program and a set of VHDL modules. This guide will detail how to install, configure and run both. It is necessary to follow the instructions in the *Integration* section, and then either of the *Installation* sections, depending on the target operating system.

1.1 Requirements

If you have a Nexys 2 board — You will need a serial cable. If your computer does not have a serial port, then you will need a USB-to-serial adapter. Here's an example: <http://amzn.com/B0007T27H8>. Any adapter will do as long as you have the right drivers for it.

If you have a Nexys 3 board — No serial cable is required, as an onboard FTDI chip provides serial emulation.

Do not integrate ratload into your RAT CPU until all the major components of the CPU are in place, and you have a good understanding of how they fit together. Understanding how the RAT architecture handles I/O and interrupts will make integration much easier.

1.2 Project Manifest

The following is a listing of each file in the `ratload` system and a brief description of the file's contents and purpose.

- `README.pdf` This guide.
- `vhdl/RS232RefComp.vhd` UART VHDL module. Required for communication with host computer.
- `vhdl/ascii_to_int.vhd` VHDL module to convert ASCII-encoded hexadecimal numbers into binary.
- `vhdl/prog_rom.vhd` Replacement `prog_rom` module for the RAT CPU.
- `vhdl/interceptor.vhd` VHDL module related to the internal operation of the `prog_rom`.
- `vhdl/prog_ram.vhd` VHDL module related to the internal operation of the `prog_rom`.
- `vhdl/real_prog_rom.vhd` VHDL module related to the internal operation of the `prog_rom`.
- `vhdl/serial_test.vhd` A special `prog_rom` module used to test the functionality of the UART.
- `asm/serial_test.asm` Assembly source of the `serial_test` VHDL module.
- `asm/ratload.asm` Assembly source for the `ratload` system.
- `bin/ratload_win/` Windows graphical version of the `ratload` program. The files in this directory are all in support of the Windows `ratload` program and will not be described individually.
- `bin/ratload_win_cli.exe` Windows command line version of the `ratload` program.
- `bin/ratload_nix` Linux and OS X command line version of the `ratload` program. Please note there is no graphical `ratload` program for Linux and OS X.



The binaries for Linux and OS X may not work for you! (Windows users can disregard this message) Your best bet if you are using a one of these OSes is to compile `ratload` from source. Instructions for doing so are available at <http://www.github.com/jhladky/ratload>.

1.3 Bug Reporting

All programs have bugs. `ratload` is beta software and is no exception. If you encounter a bug in either the `ratload` or the `winRATLoad` program, please contact the author. It is also possible (albeit significantly less likely) that you find a bug in the project VHDL modules. If you do please verify the bug via simulation and report it immediately so it can be fixed.

Since you are writing the mechanics of the CPU itself, it is possible for very odd bugs to happen, and for those bugs to interact with `ratload` in bizarre ways. Although it may seem like there is a problem with `ratload`, it has been thoroughly tested and is free of any major bugs.

1.4 Contact

If you have any questions about the project itself, or suggestions for improvement for this guide, please contact the author, at `jhladky@calpoly.edu`. This project licensed under the MIT license and the complete source code – including the \LaTeX source for this guide – is available at <http://www.github.com/jhladky/ratload>.


2 Installation

2.1 Integration on RAT CPU

Integration on the RAT CPU is split into two parts to facilitate troubleshooting. Follow the instructions in each section, and then follow instructions to verify that the actions performed in that section were successful.

2.1.1 Integration of UART

Ratload requires a UART to communicate with a host computer. The UART module is provided by Digilent and is used unmodified here.

 UART stands for “Universal Asynchronous Receiver-Transmitter”, which you may recognize as another name for a serial port. The UART can be used outside of the ratload project as well. Consider using a UART in your own final project!

1. In the Xilinx ISE Environment, go to **Project >Add Copy of Source**. Navigate to the ratload project directory (where this README is located), and then to the “vhd” folder. Select the “RS232RefComp.vhd” and “ascii_to_int.vhd” files and click **Open**. Another dialog box will pop up confirming you want to add these files. Click **OK**.

2. In the rat_wrapper.ucf (your title may differ slightly) file, add the following NETs if you have a Nexys 2 board:

```
NET "tx" LOC = P9;  
NET "rx" LOC = U6;
```

If you have a Nexys 3 board, then add these lines instead:

```
NET "tx" LOC = XX;  
NET "rx" LOC = XX;
```

3. In the architecture section of the top-level rat_wrapper module, add the following two lines to the entity declaration:

```
RXD : in STD_LOGIC;  
TXD : out STD_LOGIC;
```

4. In the same file, add the following component declarations:

```
component RS232RefComp  
  Port (  
    RXD      : in      STD_LOGIC;  
    RST      : in      STD_LOGIC := '0';  
    CLK      : in      STD_LOGIC;  
    DBIN     : in      STD_LOGIC_VECTOR(7 downto 0);  
    RD, WR   : in      STD_LOGIC;  
    RDA      : inout   STD_LOGIC;  
    TBE      : inout   STD_LOGIC := '1';  
    TXD      : out     STD_LOGIC := '1';  
    DBOUT    : out     STD_LOGIC_VECTOR(7 downto 0);  
    PE, FE   : out     STD_LOGIC;  
    OE       : out     STD_LOGIC);  
end component;  
  
component ascii_to_int  
  Port (  
    ascii_in : in      STD_LOGIC_VECTOR(7 downto 0);
```

```

        int_out : out      STD_LOGIC_VECTOR(7 downto 0));
end component;

```

5. In the same section, add the following signals:

```

signal s_d_avail      : STD_LOGIC;
signal s_d_sent       : STD_LOGIC;
signal s_d_strb       : STD_LOGIC;
signal s_d_conf       : STD_LOGIC;
signal s_db_from_rat  : STD_LOGIC_VECTOR(7 downto 0);
signal s_db_to_conv   : STD_LOGIC_VECTOR(7 downto 0);
signal s_db_to_rat    : STD_LOGIC_VECTOR(7 downto 0);

```

6. Add the following port map for the `ascii_to_int` module:

```

CONV: ASCII_TO_INT port map(
    ASCII_IN => S_DB_TO_CONV,
    INT_OUT  => S_DB_TO_RAT);

```

Data sent over the UART will be ASCII encoded, but we want it to be in hex, so we convert it as soon as it gets out of the UART module. The `S_DB_TO_CONV` signal receives an ASCII-encoded byte from the UART, and the `S_DB_TO_RAT` signal has the output in binary. You will need to add the `S_DB_TO_RAT` signal to whatever method you use to get inputs into the RAT. For example, if you have a separate input module, then send the signal into that module. If you manage your I/O in the `rat_wrapper` module, then add an `elsif` block. The `PORT_ID` for **input** from the UART is **0x0F**.

7. Add the following port map for the `RS232RefComp` module:

```

UART: RS232REFCOMP port map(
    RXD      => RXD,
    RST      => RST,
    CLK      => CLK,
    DBIN     => S_DB_FROM_RAT,
    RD       => S_D_CONF,
    WR       => S_D_STRB,
    RDA      => S_D_AVAIL,
    TBE      => S_D_SENT,
    TXD      => TXD,
    DBOUT    => S_DB_TO_CONV,
    FE       => open,
    PE       => open,
    OE       => open);

```

The following is a line-by-line breakdown of the port map:

- **RXT**: Receive port for the UART. Serial data goes into the physical UART on the Nexys board and then into this module.
- **RST**: Reset for the UART. Hook it up to the RAT's global reset. **The name of your reset pin may be different!**
- **CLK**: Clock for the UART. Hook it up to yer clock. 🕒
- **S_DB_FROM_RAT**: Send bus.

- S_D_CONF: Read strobe. We take it high to confirm that data has been received.
- S_D_STRB: Write strobe. We take it high to indicate we have data on S_DB_FROM_RAT to send.
- S_D_AVAIL: Indicates data is available to be read.
- S_D_SEND: Indicates data has been successfully sent.
- TXD: Transmit port for the UART, similar to the receive port.
- DBOUT: Receive bus
- FE, PE, OE: Error pins. “Frame Error”, “Parity Error”, and “Output Error”, respectively. We’re not concerned with checking for data errors so we leave these pins open.

8. Modify your outputs process (or module) to send data to the UART.

⚠ THIS IS A CRITICAL STEP! Many I/O devices, such as the UART, require more than just a simple data bus for proper operation. In the previous step you hooked-up several signals to the UART module, now you have to hook-up the other end! For convenience sake we will have the output process control these signals, and to make it really easy, an example output process is reproduced here.

If you use an module for your outputs, then modify the following example to fit into your module. The constant UART_OUT_ID is equal to 0x0E. **You will almost certainly need to adapt some of the signal names to fit your conventions. Make sure to do so without changing the UART signals!**

```
outputs: process (CLK) begin
    if (rising_edge(CLK)) then
        s_d_strb <= '0';
        s_d_conf <= '0';
        if (S_IO_OE = '1') then
            if (s_port_id = LEDS_ID) then
                LEDS <= s_output_port;
            elsif (s_port_id = UART_OUT_ID) then
                s_db_from_rat <= s_output_port;
                s_d_strb <= '1';
                s_d_conf <= '1';
            end if;
        end if;
    end if;
end process outputs;
```

9. Make sure that you can successfully synthesize your RAT_wrapper with the UART integrated. This is the last step in integrating the UART.

⚠ You **WILL** receive the following warning about a latch:
 Found 1-bit latch for signal <TBE>. Latches may be ...
 This is a result of a flaw in the Digilent Romania provided UART module. It is safe to ignore. Any other latch, however, is not the result of this flaw and must be fixed!

2.1.2 UART Verification

This section covers a quick test of the UART module you just integrated into your RAT CPU.

⚠ IF THE TEST IN THIS SECTION IS NOT 100% SUCCESSFUL DO NOT CONTINUE...GO BACK AND DOUBLE CHECK THAT YOU INTEGRATED THE UART PROPERLY...RATLOAD WILL NOT WORK UNLESS YOUR UART BEHAVES EXACTLY AS EXPECTED

1. Replace your current prog_rom module with the prog_rom module in the “serial_test.vhd” file, in the “vhdl” folder. Then program your Nexys board with the new bit file.
2. Hook up the serial cable to the Nexys and the host computer. If you are using Windows, follow the next step. If you are using Linux or OS X, follow the step after that.
 - a. **WINDOWS** — Run the ratload program in the “src/ratload_win/” folder. Select the proper serial device from the dropdown menu. Then go to **File >Run Serial Text**. The program will then attempt to communicate with the Nexys board via the serial cable. If the results window says “PASS”, then move on to the next section. If it says anything else, then go back and make sure you followed the integration instructions properly.
 - b. **LINUX AND OS X** — Make sure you have build the command-line ratload program correctly or can run the one provided (“src/ratload_nix”). Make sure you can identify which serial device you want to use, it will probably be something like “/dev/ttyS0” (more information on using the cli ratload is available in the “Use” section). Then run the ratload in test mode like so:

```
ratload -d /dev/<serial device> --test
```

If the program prints out “PASS”, then move on to the next section. If it says anything else, then go back and make sure you followed the integration instructions properly.

2.1.3 Integration of new prog_rom module

1. Go to **Project >Add Copy of Source**. Navigate to the ratload directory again and then to the same “vhdl” folder. Select the “prog_rom.vhd”, “prog_ram.vhd”, “real_prog_rom.vhd”, and “interceptor.vhd” files and click **Open**. Another dialog box will pop up confirming you want to add these files. Click **OK**.

⚠ This is a destructive action and will overwrite any files already in your project directory that have a name identical to the files being added e.g. any existing prog_rom.vhd file will be overwritten by the ratload prog_rom.vhd file.

2. In the architecture section of the rat_cpu module, edit the componet declaration for the prog_rom module. Add the following line:

```
TRISTATE_IN : in STD_LOGIC_VECTOR(7 downto 0)
```

3. In the same file, edit the port map decaration for the prog_rom module. Map the signal for the RAT CPU’s tristate bus into to the prog_rom module’s tristate bus. The RAT CPU’s “tristate_bus” may be called the “MULTI_BUS” in your CPU. (Regardless of its name, this is the bux that connects the ALU, the program counter, the scratch pad, and others.) That line will look similar to this:

```
tristate_in => tristate_bus_sig(7 downto 0),
```

4. Synthesize and generate a bit file for your newly integrated system. That’s it! Ratload should work on your CPU now. The next section covers how to use the ratload program.

2.2 Installation on GNU/Linux or OS X

The two operating systems (or families of operating systems) listed in the sub-section header are only three specific examples of OSes that should be able to run the ratload program. It should build and run on any POSIX-compliant OS.

To build the ratload program, cd into the ‘ratload’ directory and build the program with make. Obviously you must

have some sort of build environment set up for this to work. Mac users should have XCode tools installed or get gcc or clang via homebrew/macports/fink, etc. GNU/Linux users should install gcc via their package manager. If you're using a Ubuntu based distribution, such as Linux Mint, or Ubuntu itself, then the following will get everything you need:

```
sudo apt-get install build-essential
```

After building the binary, there is no installation step. Simply copy it to some convenient location (such as a personal bin folder).

2.3 Installation on Windows

Windows users must use the winRATLoad program

Copy the “winRATLoad” folder to somewhere useful for you. The winRATLoad executable must remain in the folder to function properly. The program is otherwise self contained and no other installation is necessary.

2.4 Serial Configuration

2.4.1 Nexys 2

If you have a serial port on your computer, congratulations, you are using a computer from the 20th century! Your OS almost certainly already has installed drivers to use this port, so you don't need to do anything else.

If you don't have a serial port, then you need to obtain a USB-to-serial adapter cable. If you're on Windows, the manufacturer of the cable probably provided drivers for you to install along with the cable itself, make sure you install them. If you're using GNU/Linux or OS X, google around to try to find the right driver.

Once you think you've installed the driver then you simply need to verify that the OS can see it. In Windows 7, go to the Start menu and right click Computer, and then click manage. Look for the serial device in the device management section of the console. In GNU/Linux or OS X, list the contents of the `/dev` folder. In GNU/Linux your serial adapter will be called something similar to “ttyS0”. In OS X, it will look like “tty.usb0”, or similar.

2.4.2 Nexys 3

If you have a serial port on your computer, it doesn't matter, you can't use it! The Nexys 3 doesn't have a physical serial port, but instead emulates one with an FTDI chip. The USB cable you use to power and program the board also functions as a serial cable when it needs to. This means you need to install an FTDI driver. FTDI drivers for OS X and Windows 7 are included in the project folder. Install them and be glad you don't have to deal with USB-to-serial cable drivers!

3 Use

Up until now, you’ve probably followed this pattern when developing your assembly language programs:

1. Run your assembly repeatedly in the ratsim program until it produces a prog_rom.vhd file and seems to be relatively bug free.
2. Add the new prog_rom.vhd file to your project, replacing the old one.
3. Resynthesize the entire project and reprogram the bit file onto your Nexys board.
4. Repeat ad infinitum.

The steps you have just followed to integrate the vhd files into your RAT CPU, and to install the ratload program onto your computer, will dramatically change this pattern. From now on you do not need to resynthesize your project when ratsim generates a new prog_rom.vhd file, nor is it necessary to copy that new file into your project. In fact, **making any more changes to the prog_rom.vhd file in your project directory will break ratload.**

The following is a general overview of the new pattern you need to follow in order to use ratload properly:

1. Program your Nexys board with the generated bit file.
2. Without disconnecting anything else, connect the Nexys 2 board to your computer via the serial cable. **If you have a Nexys 3 board, ignore this step.**
3. Start either the winRATLoad or the ratload program, depending on your OS. Select the proper serial device and prog_rom.vhd file to read.
4. The program will then communicate with the Nexys board and send the prog_rom.vhd to your RAT CPU via the serial connection. Once the program displays a success message, the serial cable (again, only if you have a Nexys 2) can be disconnected and the board used normally. You can treat the program running on the board like it was synthesized with the system using the previous pattern.
5. To send a new prog_rom.vhd file you must power cycle the Nexys board (If you programmed your bit file into volatile memory, then you’ll need to reprogram it as well.) Then repeat this procedure.

3.1 Using winRATLoad (Windows)

3.2 Using ratload (GNU/Linux or OS X)

ratload takes exactly two arguments: “-d” to specify a serial device and “-f” to specify a prog_rom.vhd file to parse. You’ll probably have to run it as root in order to access the serial device. The following is an explanation of all the errors messages ratload can produce:

- “Opening prog_rom failed”: Ratload could not open your prog_rom.vhd file. Perhaps it couldn’t find it, or it didn’t have permission.
- “Opening serial device failed”: Ratload could not open the serial device you specified. Perhaps you specified it incorrectly, or ratload does not have permission to access it.
- “Invalid prog_rom.vhd file, exiting.”: Ratload was able to find and open the file you specified, but it couldn’t parse it. Ratload expects the prog_rom.vhd to be structured in a very specific way. Because this file is auto-generated by the ratsim program, this is not a problem. Make sure you are specifying the exact prog_rom.vhd file that ratsim generates. If you continue to receive this error, try generating the prog_rom.vhd file again.
- “Serial Configuration Failed”: Ratload was able to find and open the device you specified, but when it failed to configure it. It is possible but extremely unlikely that you have a serial device that does not support the proper settings. It is much more like that you specified a valid but incorrect device.

- “Error communicating with Nexys2 board.”: Ratload was able to open and parse the file you specified, and was able to open and configure the serial device you specified, but it received no or incorrect data from the Nexys board. Program the “reference_rat_wrapper.bit” file onto your Nexys board and try to send data to it with ratload. If that works, then you have misconfigured your RAT CPU, and you need to return to the integration section and make sure you followed those steps correctly.
- “Too many [few] arguments.”: You specified the arguments to ratload incorrectly.
- “Option not supported. Only -f and -d supported.”: You specified the arguments to ratload incorrectly.