`ratload` Installation and Use

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

February 19, 2015

1 Introduction **2**
1.1 Requirements . . . 2
1.2 Project Manifest . . . 2
1.3 Bug Reporting . . . 3
1.4 Contact . . . 3

2 Adding the UART **4**
2.1 Integration on to RAT CPU . . . 4
2.2 Verification . . . 6
2.3 Troubleshooting . . . 7
2.3.1 Obtaining a serial console . . . 7
2.3.2 Configuring the proper serial console settings . . . 7
2.3.3 Using the serial console to troubleshoot the UART . . . 7

3 Adding the New Prog_Rom module **7**

4 Installing the Ratload Program **10**
4.1 Graphical ratload (Windows only) . . . 10
4.2 Building from source . . . 10

5 Use **11**
5.1 Identifying your Serial Device . . . 11
5.2 Using winRATLoad (Windows) . . . 11
5.3 Using ratload (GNU/Linux or OS X) . . . 11

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 necesssary to follow the instructions in both *Integration* sections, 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 hexidecimal 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.exe` Windows command-line version of the ratload program.
- `bin/ratload_nix` Linux command-line version of the ratload program.
- `bin/ratload_osx` 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 building ratload from source are available in Section 4.2.

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…However `ratload` 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 Adding the UART

Before you can use `ratload`, you have to add a UART module to your CPU, which is required to communicate with the host computer. Adding the UART is similar to other I/O devices you have encountered such as the 7-segment display driver and the VGA buffer.

☞ 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!

2.1 Integration on to RAT CPU

This section will cover adding the UART module to the RAT CPU and hooking up all the pins to the right places. The UART module is provided by Digilent and is used unmodified here.

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 file (your title may differ slightly), add the following NETs if you have a Nexys 2 board:
```
NET "TXD" LOC = P9;
NET "RXD" LOC = U6;
```
   If you have a Nexys 3 board, then add these lines instead:
```
NET "TXD" LOC = XX;
NET "RXD" 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.

7. Add the `S_DB_TO_RAT` signal to whatever method you use to get intputs 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**.

8. 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:

   - `RXD`: 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.

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

⚠ **THIS IS A CRITICAL STEP!** I/O devices like the UART require more than just a simple data bus for proper operation. In the previous step you hooked-up several control 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.

You may need to adapt some of the signal names to fit your conventions. **Make sure to do so without changing the UART signals!**

```
CONSTANT UART_OUT_ID   : STD_LOGIC_VECTOR(7 downto 0) := x"0E";

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;
```

10. 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!

2.2 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 have integrated the UART properly. If the UART continues not to work, go to the "Troubleshooting" section. **`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. This step involves running the ratload program. For more information about installing and running the ratload program see the “Installation” and “Use” sections.
   a. **Graphical version (Windows only) —** Select the proper serial device from the dropdown menu. Then go to **File >Run Serial Test**. 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. **Command-line version —** Run ratload in test mode like so:

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

      If the program prints out “PASS”, then you can skip the “Troubleshooting” section. If it says anything else, then go back and make sure you followed the integration instructions properly.

2.3 Troubleshooting

The easiest way to troubleshoot the UART module is to break out an actual serial console and see what it’s sending to the computer. So that’s what we’re going to do. You will need to know what your serial device is called on your computer. To do so see Section 5.1.

2.3.1 Obtaining a serial console

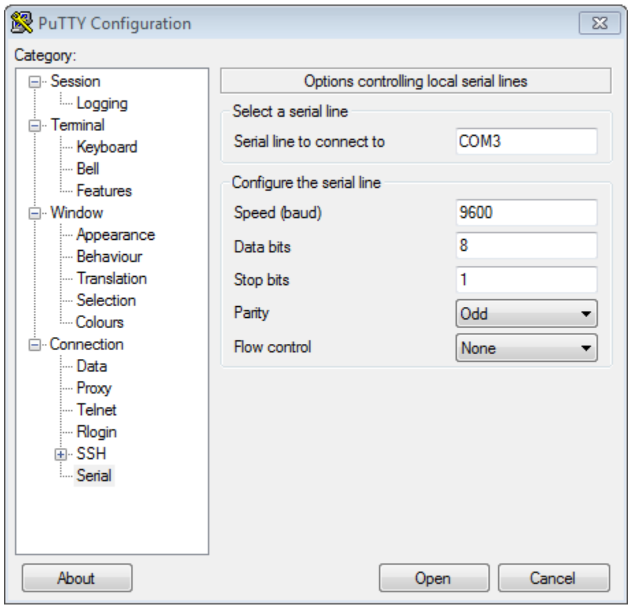
- **Windows users —** A good serial console for Windows is putty. Download putty from here:
  `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`
- **Linux users —** Install minicom with your package manager. If you are using a debian-based disto that looks like: `sudo apt-get install minicom`.
- **OS X users —** Install minicom with homebrew, macports, whatever. That will look like: `brew install minicom`.

2.3.2 Configuring the proper serial console settings

- **Windows users —** Run putty. From the **Connection Type** options, select the **Serial** radio button. Enter the COM port of your serial device in the **Serial Line** box. Before you can connect on the serial line you need to make sure it has the correct settings. Make your putty settings look like they do in Figure 1. Remember that your COM port may differ. After you have entered the correct settings click **Open**.
- **Linux and OS X users —** Start minicom from the terminal in config mode by using the `-s` flag, like so: `minicom -s`. Use your keyboard to navigate the configuration menu that pops up. Select the **Serial Port Setup** option. Make your minicom settings look like they do in Figure 2. Remember that your serial device and “Lockfile Location” may differ (you can ignore “Lockfile Location”, actually). After you gave entered the correct settings hit escape until minicom dumps you out to the serial console window.

2.3.3 Using the serial console to troubleshoot the UART

Once you have your serial console up and running flash your Nexys board with your bit file that has the serial test program on it. (Make sure you’ve connected the Nexys board to your computer with the serial cable).



Options controlling local serial lines

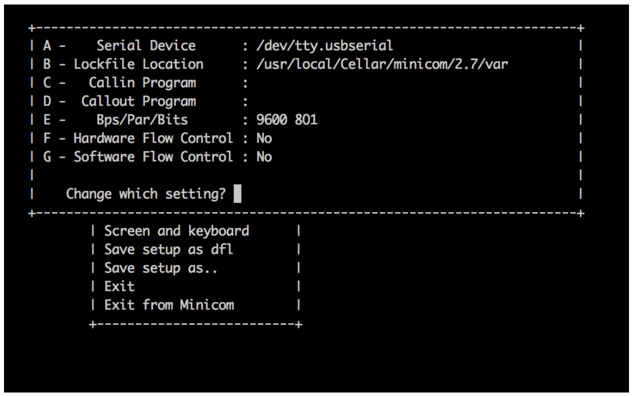
Select a serial line

Serial line to connect to	COM3

Configure the serial line

Speed (baud)	9600
Data bits	8
Stop bits	1
Parity	Odd
Flow control	None

Figure 1: The correct settings to use with Putty



```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/tty.usbserial                        |
| B - Lockfile Location     : /usr/local/Cellar/minicom/2.7/var         |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 9600 8O1                                  |
| F - Hardware Flow Control : No                                          |
| G - Software Flow Control : No                                          |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
        | Screen and keyboard      |
        | Save setup as dfl        |
        | Save setup as..          |
        | Exit                     |
        | Exit from Minicom        |
        +--------------------------+
```

Figure 2: The correct settings to use with minicom

3 Adding the New Prog_Rom module

Once you have added the UART and verified that it is functioning correctly, the remaining steps are simple.

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 sections cover installing and using the ratload program.

4 Installing the Ratload Program

Installation is really easy. For all platforms the command-line ratload binary is self-contained. Installation consists of copying the program to a location convenient to you. Make sure to choose the binary corresponding to your platform. Linux users need "ratload_nix", OS X users need "ratload_osx", and Windows users need "ratload_win.exe". Instructions for using the grapical version of ratload (Windows only) and for building from source and included below.

4.1 Graphical ratload (Windows only)

The graphical version of the ratload program is located in the folder "ratload_win". Copy the entire folder somewhere useful to you. The graphical program will not work without the container folder. There is no other installation step.

4.2 Building from source

The Windows versions of ratload contain statically linked binaries and thus are unlikely to malfunction across different versions of Windows. The Linux and OS X versions are not statically linked and if you have problems running them you should compile ratload from source.

Ratload has no dependencies other than the libc and should build and run on any POSIX-compliant OS. The source is available at `http://github.com/jhladky/ratload`, in the `nix` folder. Clone the repo and build the program with `make`.

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

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

If you're using some other distro I'm not going to help you because you know what you're doing. After building the binary, follow the above instructions for your relevant OS.

5 Use

5.1 Identifying your Serial Device

- **Windows users —** Run the graphical ratload program. The main window of the program contains a dropdown which lists your available serial devices. If you are using a USB-to-serial adapter the easiest way to be totally sure of the correct COM port number is to unplug the adapter and run the graphical ratload program. While the adapter is still unplugged, note the adapters listed. Then plug in the adapter and click **File > Refresh Serial Devices**. The COM port that appears in the dropdown is the COM port of the adapter.
- **Linux and OS X users —** 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.

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.

5.2 Using winRATLoad (Windows)

5.3 Using ratload (GNU/Linux or OS X)

ratload takes exactly two arguments: "-d" to specify a serial sevice 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 exlpanation 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 wih 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.