The University of Kansas

Technical Report

Automated RF Testing of the KU Agile Radio 2.0 Using LabView

Brian Cordill

ITTC-FY2008-TR-31620-07

February 2008

Project Sponsor:
National Science Foundation
Computer and Information Science and
Engineering Directorate

December 18, 2006

FROM: Brian Cordill

TO: Dr. Minden

SUBJECT: Automated RF Testing of the Agile Radio 2.0 using LabView

1. Statement of Purpose

The objective of this project is to define, construct and carryout testing of the RF front end of the Agile Radio ver. 2.0. In this initial stage the focus will be on characterizing the gain ripple across the transmitter bandwidth. This bandwidth stretches from 5.25 GHz to 5.85 GHz with channels spaced every 4 MHz. With approximately 150 channels and 6 bits of variable gain, the need for an automated testing regimen is readily apparent. National Instrument's LabVIEW is a natural choice with its capability to configure test equipment and capture data.

2. Theory and Design

2.1 Theory –The two major subsystems within the transmitter that the current stage of testing must configure are the transmitter gain, and frequency. The gain for the transmitter is set by issuing commands to an MC68HC08 microcontroller which in turn sets the control voltage for a variable gain amplifier, VGA, using a 6-bit DAC. The relevant section of the radio block diagram is shown in Figure 1, the micro controller inputs can be seen as blue lines entering the top of the diagram. The 6-bit DAC produces a voltage between 0 and +3.3 volts based on its SPI interface with the microcontroller. This voltage is then used as the control voltage for the variable gain amplifier. A typical gain vs. control voltage plot, taken from the data sheet for this amplifier is given in Figure 2. An important note here is that this amplifier is located in the IF section of the transmitter chain and is operating at approximately 2 GHz. It is primarily this amplifiers frequency response that will dictate the total output gain of the transmitter.

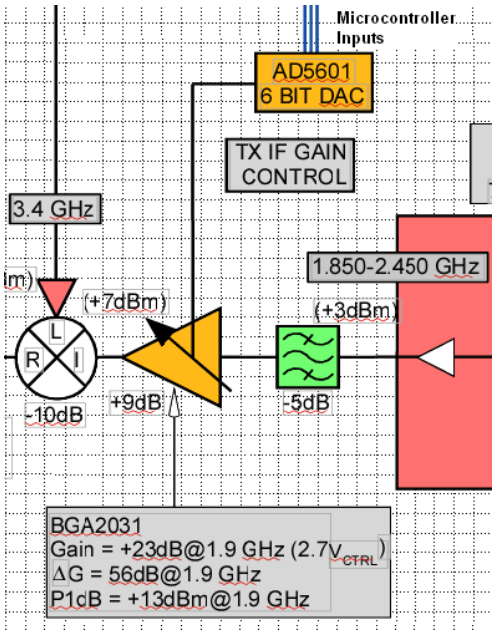
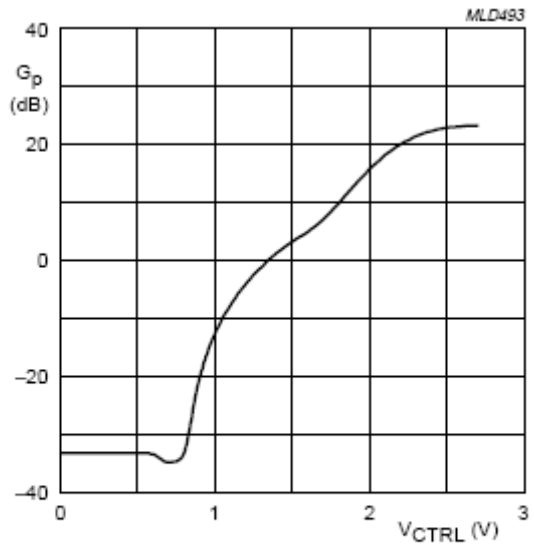


Figure 1: Tx Gain Control



V_S = 3 V; P_D = –14 dBm; f = 1.9 GHz.

Figure 2: Control Voltage vs. Gain of the Tx Variable Gain Amplifier

The transmitter's frequency is set by issuing commands to the same microcontroller used to configure the transmitter gain. In this case one of two local oscillators is powered on and set to the desired IF frequency. LOA operates from 1.85 - 2.15 GHz while LOB operates from 2.15 - 2.45 GHz, the IF frequency is latter mixed up an additional 3.4 GHz to its final transmit frequency. The block diagram of this section of the radio is shown in Figure 3.

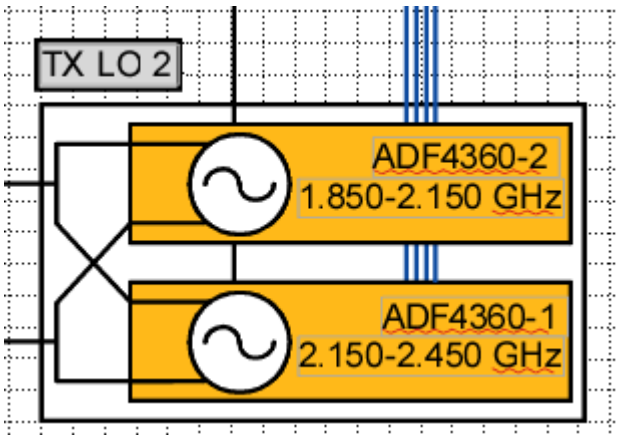


Figure 3: Tx Frequency Control

2.2 Design – Automated testing must be able to do the following:

1) Issues commands to configure the transmitter's frequency & gain.
2) Configure the digital board's FPGA to generate a sine wave to act as the baseband data signal.
3) Configure lab test equipment & capture data points
4) Process raw data points into more practical test data

Communication with the Agile Radio is performed though a secure shell terminal, and the rfControl configuration command. The rfControl command interfaces with the microcontroller on the RF board to configure the board's settings. The process of sending a command begins by assembling the command line in LabVIEW, writing it to an external command file, and then calling an external ssh client, in this case Putty, to send the command file. Th virtual instrument, vi, build to accomplish this is shown in Figure 4.

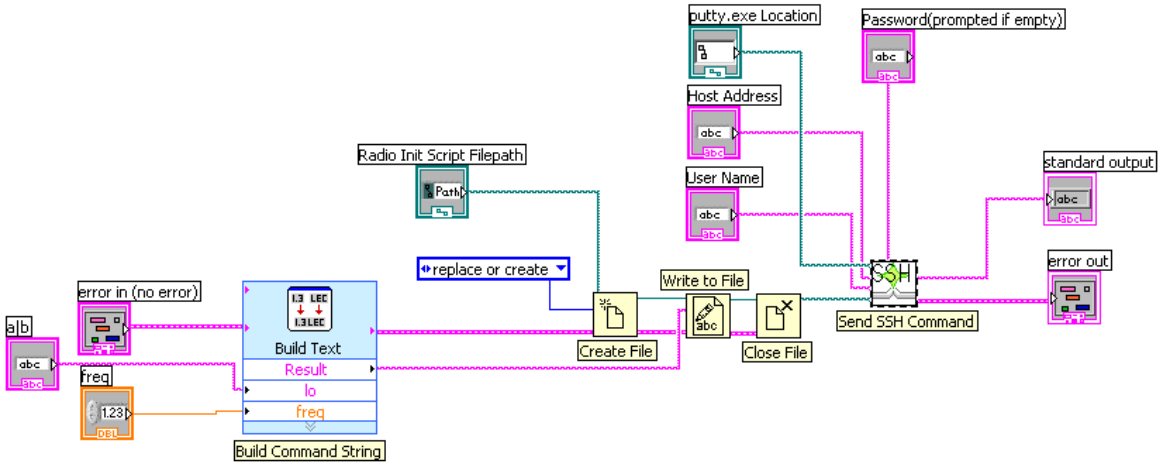


Figure 4: SSH Command VI

Configuration of the FPGA is accomplished by a combination of the fpgaCnfg and fpgaRW commands. These commands are sent in an identical manner to the RF configuration commands. The fpgaCnfg command loads the kuar_test_multicarrier_01.bit file into the FPGA. This configuration can generate up to four unmodulated sign waves. The properties of each sign wave are contained in a control register established by the multicarrier bit-file. The fpgaRW command can access these control registers and modify the content.

LabVIEW has build-in modules for connection to, configuration of, and take measurement from a large number of lab instruments. In this stage of testing an HP 8593E spectrum analyzer is used to measure the power spectrum density of the transmitter from the Agile Radio. LabVIEW has an existing module to control this instrument and no additional programming was needed. Each data capture consists of 401 power spectrum density values, expressed in dB. Each capture appears on a separate line of the output data file. After all the measurements at a given frequency are taken an information line is inserted into the data file providing frequency indexing, and control

voltage level information, this also serves to separate data block for one frequency from one another.

Each line of the output file created by LabVIEW is a space-separated list containing 401 points representing the power spectrum density captured by the spectrum analyzer. Before this file can be imported into Matlab its syntax must be amended. Matlab can read space-separated list but it needs to of the form:
<variable> = [<data1> <data2> … <dataX>];

A simple Unix awk script provides the necessary text manipulation to add the required boiler plate. The script reads in the raw data file, adds a unique variable name, inserts the necessary brackets and prints the new line to a new output file. The full scrip can be fond in the appendix. The raw data is now ready to be imported into Matlab for further processing.

The first task Matlab performs is correlate captures from each frequency into a single matrix with an identifiable name. This matrix contains a row for every capture made at that frequency, and are ordered from the lowest gain-voltage setting to the highest. This process of naming and correlating is accomplished by the Rename.m Matlab script, see appendix. Rename.m starts by generating a list of unique variable names based on the IF frequency, and then copying the data capture for that frequency into each row of the new variable name. Rename.m also generates a set of information variables that hold the transmitter frequency, and control voltage. With the data correlated into easier to understand variable matrixes a whole range of processing can be performed with much greater ease then before.

3. Laboratory Evaluation

3.1 Configuration –

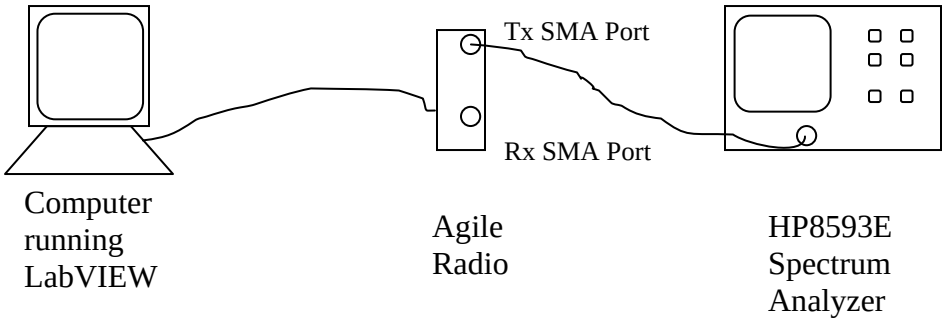


Figure 5: Test Equipment Setup

3.2 Procedure – Connect the test equipment as seen above. Fill in the require fields in the configuration screen, Figure 6. It may also be necessary to fill in the Radio Init Scrip

Filepath, and the Gain Script Filepath in the “Front End.vi”, depending on what test version is being used. Once the LabVIEW test is complete run the awk script and then the Matlab script.

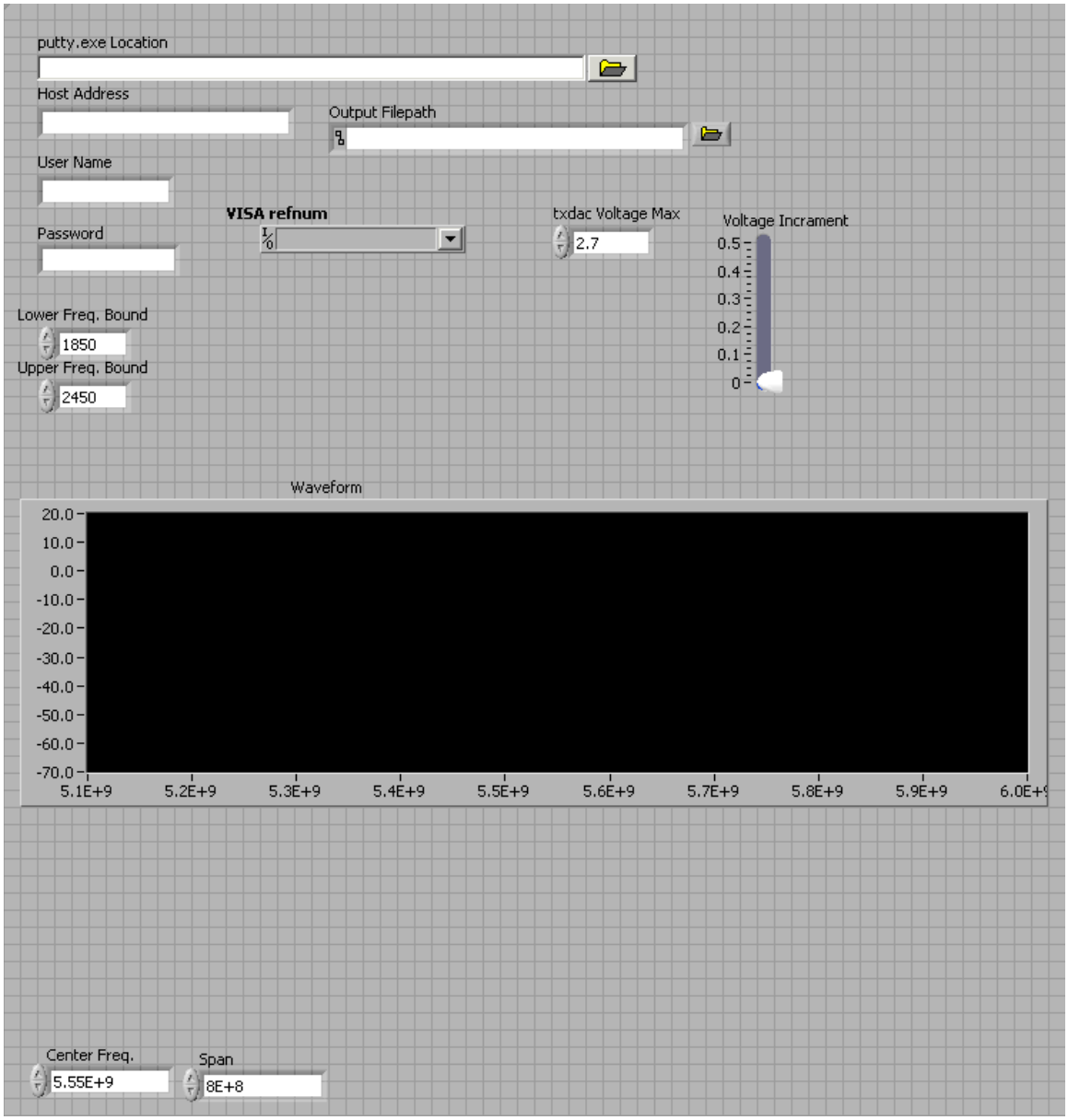


Figure 6: LabVIEW configuration Screen

4. Evaluation of Test Data

4.1 Data – Figures 7 & 8 are different cuts of the same data. In Figure 7 each line represents the output power across control voltage at a give frequency. In Figure 8 each line represents the output power of a given control voltage across the transmitter bandwidth.

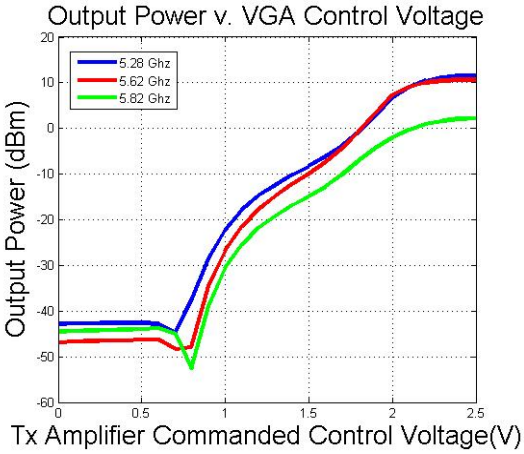


Figure 7: Output Power v. Voltage-Gain Amplifier Control Voltage

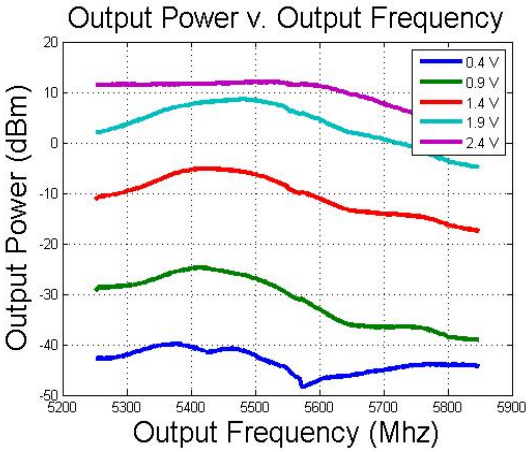


Figure 8: Output Power v. Output Frequency

4.2 Interpretation – There is a strong similarity between the measured results in Figure 7 and the VGA data sheet plot seen in Figure 2. While the test data is for a higher frequency then that shown in the data sheet, this gives a strong indication that the VGA is operating as intended.

Figure 8 shows the ripple in the output power across the transmitter's bandwidth. From this it is apparent that the VGA's gain falls off in the upper half of the radio's range. This fall off is as much as 10 dB in some cases. This can also be seen in Figure 7, the 5.82 GHz signal is almost 10 dB lower then the 5.28 & 5.62 GHz signal.

4.3 Conclusions – National Instruments LabVIEW adapts well the automated testing of the Agile Radio's transmitter. And with the output power curves at individual frequencies closely matching gain curves provided in the amplifiers data sheet, the correlated data across the transmitters bandwidth can be read with a measure of confidence. The output ripple across the transmitter's bandwidth varied with the VGA's control voltage, but in some cases reached as much as 15 dB.

5. Appendix *(code)*

```
#!/bin/sh

# Author: Brian Cordill
# Date: May 25, 2006

# This script uses awk to process data into a mablab readable format.
# Raw data must be space seporated, and may be of any length.
# Processed data will have a variable name assignement of the
# form x#=[ <DATA> ];
# Where x# is x1, x2, x3,... depending on which line the data
# comes from.


if [ $# -ne 2 ]
then
   echo "Error in $0 - Invalid Argument Count"
   echo "Syntax: $0 input_file output_file:"
   exit
fi

awk '{print "x" NR "=[",$0,"];"}' < $1 > $2
```

```
% Renames resulting variables from the "Frequency Walker.vi" agile radio Tx
% test.  Results in two variables for each tested frequency:
% freq_xxxx_data, and freq_xxxx_info.  Data contains the gain test results
% for that frequency.  Info contains frequency and indexing information in
% the format: [<starting freq.> <freq. incrament> <IF freq.> <starting voltage>
% <voltage incrament> <ending voltage>]
number_of_variables=length(who('x*'));
real_names=char(who('x*'));    % Character string list of variable names
N=size(real_names); % will need to know the length of name string

variable_index=[]; %real_names index of info variables
freq_list=[];
array_list=[];
for i=2:number_of_variables     % Generate new names for the info variable,
                                % freq_####_info and setup new array names.
    if(length(eval(real_names(i,[1:N(2)])))==6)
        variable_index=[variable_index,i]; % Save the index for later
        P=eval(real_names(i,[1:N(2)])); % Grab the values from the old name
        new_name=genvarname(['freq_',num2str(P(3)),'_info']); %Create a new info name
        new_array=genvarname(['freq_',num2str(P(3)),'_data']);%Create a new data name
        freq_list=[freq_list;new_name]; % Store that info name in a list for later
        array_list=[array_list;new_array];% Store the data name in a list for later
        eval([new_name '=P;']); % Asign new name the old value,
        eval([new_array '=[];']);
%        eval(['clear ' real_names(i,[1:N(2)])]) % Clear the old name from workspace
    end
end
% Copy values from old "x" names to new "freq_####_data" names
for i=1:length(variable_index)
```

```
    current_array=array_list(i,[1:14]) %Get next new var name
    if i==1 %First Case
        for k=2:variable_index(i)-1
            source_array=real_names(k,[1:N(2)]); %Get next old name
            eval([current_array '=[' current_array ';' source_array '];']) %Copy
values
        end
    elseif i~=length(variable_index) %Middle Cases
        for k=variable_index(i-1)+1:variable_index(i)-1
            source_array=real_names(k,[1:N(2)]);
            eval([current_array '=[' current_array ';' source_array '];']);
        end
    else %End Case
        for k=variable_index(i-1)+1:variable_index(length(variable_index))-1
            source_array=real_names(k,[1:N(2)]);
            eval([current_array '=[' current_array ';' source_array '];']);
        end
    end
end
```