ARRL 100 YEARS

QEX

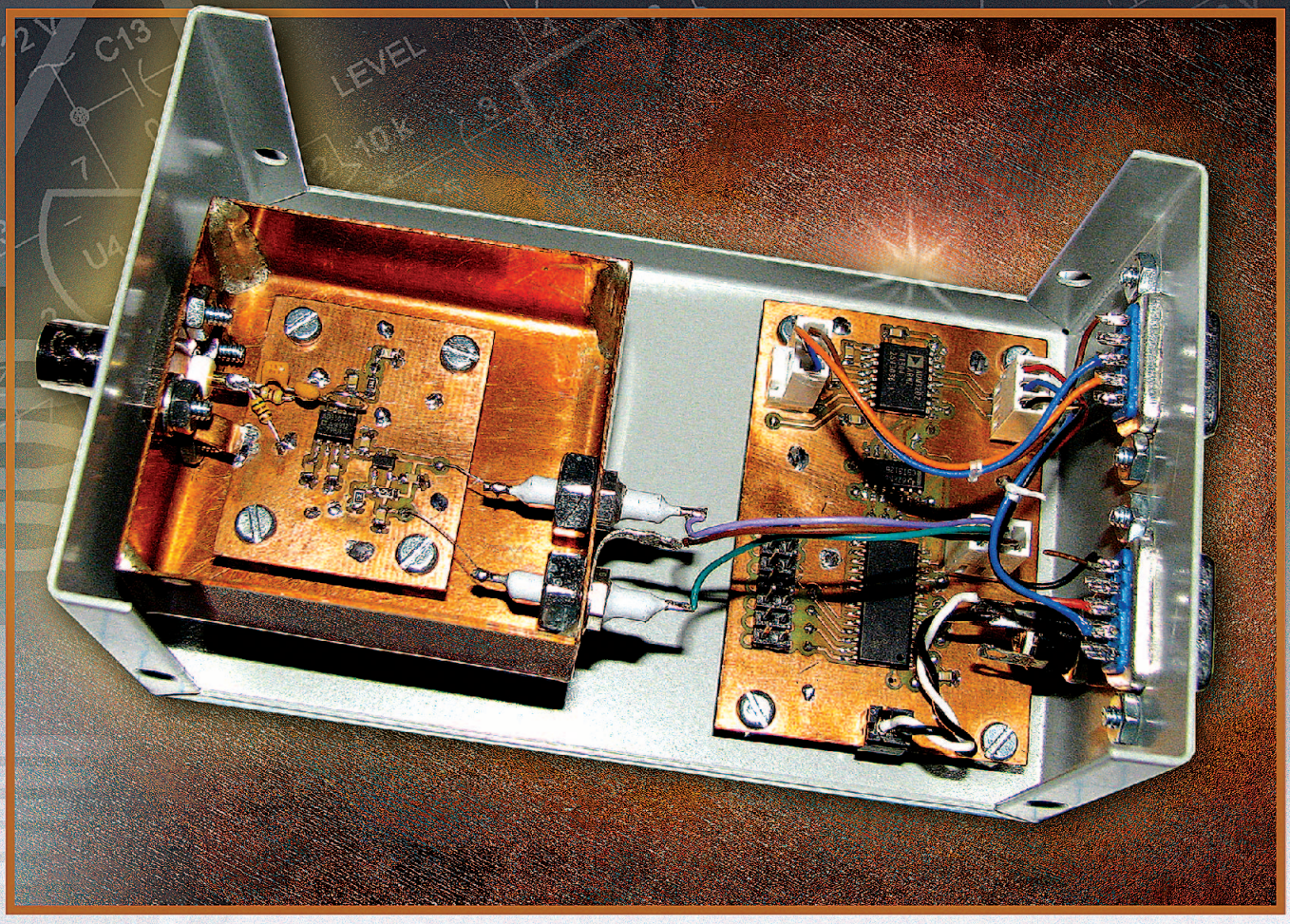
$5

July/August 2014

www.arrl.org

A Forum for Communications Experimenters

Issue No. 285



AA7VM presents "An RF Filter Evaluation Tool" for your test bench. An RF detector board (left) and a microprocessor controller board (right) form the heart of this measuring system.

ARRL The national association for AMATEUR RADIO™
225 Main Street
Newington, CT USA 06111-1494

HF/50MHz ALL MODE TRANSCEIVER TS-590

Nothing But Performance

The TS-590S

Kenwood has essentially redefined HF performance with the TS-590S compact HF transceiver. The TS-590S RX section sports IMD (intermodulation distortion) characteristics that are on par with those "top of the line" transceivers, not to mention having the best dynamic range in its class when handling unwanted, adjacent off-frequency signals.*

- **HF-50MHz 100W**
- **Digital IF Filters**
- **Built-in Antenna Tuner**
- **Advanced DSP from the IF stage forward**
- **500Hz and 2.7KHz roofing filters included**
- **Heavy duty TX section**

- **2 Color LCD**

KENWOOD

Customer Support: (310) 639-4200
Fax: (310) 537-8235

Scan with your phone to download TS-590S brochure.

www.kenwoodusa.com

ISO9001 Registered
Professional Systems Business Group
JVC KENWOOD Corporation

ADS#36812

*** For 1.8/3.5/7/14/21 MHz Amateur bands, when receiving in CW/FSK/SSB modes, down conversion is automatically selected if the final passband is 2.7KHz or less.**

QEX

QEX (ISSN: 0886-8093) is published bimonthly in January, March, May, July, September, and November by the American Radio Relay League, 225 Main Street, Newington, CT 06111-1494. Periodicals postage paid at Hartford, CT and at additional mailing offices.

POSTMASTER: Send address changes to: QEX, 225 Main St, Newington, CT 06111-1494
Issue No 285

Harold Kramer, WJ1B
Publisher

Larry Wolfgang, WR1B
Editor

Lori Weinberg, KB1EIB
Assistant Editor

Zack Lau, W1VT
Ray Mack, W5IFS
Contributing Editors

Production Department

Steve Ford, WB8IMY
Publications Manager

Michelle Bloom, WB1ENT
Production Supervisor

Sue Fagan, KB1OKW
Graphic Design Supervisor

David Pingree, N1NAS
Senior Technical Illustrator

Brian Washing
Technical Illustrator

Advertising Information Contact:

Janet L. Rocco, W1JLR
Business Services
860-594-0203 – Direct
800-243-7768 – ARRL
860-594-4285 – Fax

Circulation Department

Cathy Stepina, *QEX Circulation*

Offices

225 Main St, Newington, CT 06111-1494 USA
Telephone: 860-594-0200
Fax: 860-594-0259 (24 hour direct line)
e-mail: **qex@arrl.org**

Subscription rate for 6 issues:

In the US: ARRL Member $24, nonmember $36;

US by First Class Mail:
ARRL member $37, nonmember $49;

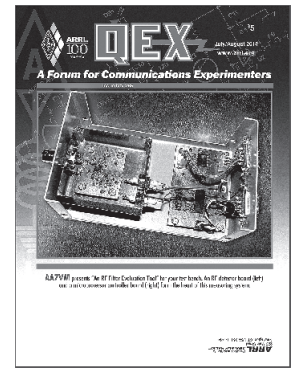
International and Canada by Airmail: ARRL member $31, nonmember $43;

Members are asked to include their membership control number or a label from their QST when applying.

In order to ensure prompt delivery, we ask that you periodically check the address information on your mailing label. If you find any inaccuracies, please contact the Circulation Department immediately. Thank you for your assistance.

About the Cover

Gary Richardson, AA7VM, designed "An RF Filter Evaluation Tool" that will find plenty of use on your test bench if you build or adjust RF filters. A microprocessor controller board sends command signals to a signal generator, which feeds the test signal through the filter and to the input of an RF detector board. The microprocessor board then reads the RF power measurements from the detector and sends them to a computer.



In This Issue

Features

Index of Advertisers

The American Radio Relay League

The American Radio Relay League, Inc, is a noncommercial association of radio amateurs, organized for the promotion of interest in Amateur Radio communication and experimentation, for the establishment of networks to provide communications in the event of disasters or other emergencies, for the advancement of the radio art and of the public welfare, for the representation of the radio amateur in legislative matters, and for the maintenance of fraternalism and a high standard of conduct.

ARRL is an incorporated association without capital stock chartered under the laws of the state of Connecticut, and is an exempt organization under Section 501(c)(3) of the Internal Revenue Code of 1986. Its affairs are governed by a Board of Directors, whose voting members are elected every three years by the general membership. The officers are elected or appointed by the Directors. The League is noncommercial, and no one who could gain financially from the shaping of its affairs is eligible for membership on its Board.

"Of, by, and for the radio amateur," ARRL numbers within its ranks the vast majority of active amateurs in the nation and has a proud history of achievement as the standard-bearer in amateur affairs.

A *bona fide* interest in Amateur Radio is the only essential qualification of membership; an Amateur Radio license is not a prerequisite, although full voting membership is granted only to licensed amateurs in the US.

Membership inquiries and general correspondence should be addressed to the administrative headquarters:

ARRL
225 Main Street
Newington, CT 06111 USA
Telephone: 860-594-0200
FAX: 860-594-0259 (24-hour direct line)

Officers

President: KAY C. CRAIGIE, N3KN
570 Brush Mountain Rd, Blacksburg, VA 24060

Chief Executive Officer: DAVID SUMNER, K1ZZ

The purpose of *QEX* is to:

1) provide a medium for the exchange of ideas and information among Amateur Radio experimenters,

2) document advanced technical work in the Amateur Radio field, and

3) support efforts to advance the state of the Amateur Radio art.

All correspondence concerning *QEX* should be addressed to the American Radio Relay League, 225 Main Street, Newington, CT 06111 USA. Envelopes containing manuscripts and letters for publication in *QEX* should be marked Editor, *QEX*.

Both theoretical and practical technical articles are welcomed. Manuscripts should be submitted in word-processor format, if possible. We can redraw any figures as long as their content is clear. Photos should be glossy, color or black-and-white prints of at least the size they are to appear in *QEX* or high-resolution digital images (300 dots per inch or higher at the printed size). Further information for authors can be found on the Web at **www.arrl.org/qex/** or by e-mail to **qex@arrl.org**.

Larry Wolfgang, WR1B

Empirical Outlook

Sharing Ideas

I have occasionally used this space to encourage readers to document their projects and share their ideas by writing for *QEX*. After all, the subtitle of our magazine is *A Forum For Communications Experimenters*, and *Forum* implies shared information. You may think that this is a plea for you to submit an article for publication. It is certainly true that we need a lot of good article submissions to fill the pages of *QEX*. So yes, it is.

There are other reasons to write about your projects than just publishing them in *QEX*, though. You are going to need some notes or documentation for later reference. I can almost guarantee that some time later you will have to troubleshoot the device, or you may even want to modify it to make use of a new IC or circuit idea. Sure some brief notes in a notebook or computer file may help you remember the details, but how much easier would it be if you could read the description you wrote for publication? With the extra care you will have to put into documenting the project for others, you will have a more complete record of your work.

There are other Amateur Radio operators who are also trying to do the same thing you have been working on, or something very similar. At first, everyone has to "discover" all about the project for themselves. That can be fun, and even provide it's own reward, but sometimes it is nice to know what others have been doing, rather than having to reinvent the wheel. There are also some experimenters who aren't engineers, and may not be able to invent the wheel, but who would really enjoy building a wheel similar to yours, and learning how it works. With your help, they will discover some of the same thrills you have experienced.

Here is one example that I would like to share. It's a project I have recently become interested in, and have begun to "play" with. We know that there are groups of hams who have taken off-the-shelf wireless routers and installed new firmware to create Amateur Radio networks. These have been called high speed multi-media (HSMM) mesh networks, although the more recent term seems to be broadband-hamnet. A number of these reprogrammed routers can be deployed to create a network in a remote area or during an emergency communications event. There have been a few *QST* articles about the use of such networks by ARES groups or for other applications. Some clubs have used these mesh networks to link their logging computers at Field Day, for example. There is quite a bit of information on various websites, and the one that I have found that seems to have the most information is **www.broadband-hamnet.org**.

By searching through the documentation on this website I have been able to find the list of routers that will work with the new firmware. Then I found a pair of LinkSysWRT54G routers that met the criteria at a tag sale, for $3 each. More searching brought me to a set of instructions for flashing the new firmware into these routers. Great! Now what? Slowly I am digging through the information to learn how I might link computers, create an access point to my little broadband-hamnet network and eventually be able to use it in some way. At this point I have one router connected via LAN cable to my Raspberry Pi, and the other one connected via LAN cable to my iMac. How can I connect to my network with my iPhone or my wife's *Windows* laptop via a WiFi link? There must be a way, I think. More searching.

My point is that there is a lot of information, but it is not necessarily the clearest documentation for someone who knows nothing about creating and using these networks. The Technical Editorial Staff at ARRL Headquarters has discussed our desire to have an article or series of articles for *QST* and/or *QEX* about these routers and networks. At first we had the mistaken impression that the routers that could be flashed with this new firmware were obsolete models that would only be available from a junkbox or lucky find at a tag sale. Publishing an article about something that requires you to find some obsolete hardware is not generally the best idea. We have since learned that there is at least one model (the LinkSys WRT54GL) that is a currently available model. We have also learned that the movers and shakers in this project have been working on creating firmware for some other current production routers.

So there is a need for an article that gives the basic details of what to look for in a router, including sources for new hardware, a description of how to flash the new firmware into the router and how to configure the mesh node. There is a need for a description of how to connect to the network with either a wired LAN connection or a WiFi connection, and how to then pass data around the network to and from the various computers on the network. An article or series of articles about these networks would serve to document the details in a publication for those new to the topic. The more people who become involved in building these nodes and adding them to the overall network, the more we can do with this technology.

I haven't even touched on the possibilities of adding high-gain external antennas and even power amplifiers to the routers. There are many possibilities when we continue experimenting with the networks as Amateur Radio operators.

This is just one example. There are lots of other projects that hams are experimenting with. Help share the wealth!

Gary Richardson, AA7VM

PO Box 228, Marblemount, WA 98267; **aa7vm@arrl.net**

An RF Filter Evaluation Tool

Here is a microprocessor controlled system to measure filter frequency response.

Unless you have a spectrum analyzer, determining the response of an RF filter is rather tedious — adjust the signal generator, measure the output, record the value and repeat until sufficient data has been collected. Then you must convert the values to decibels and make a plot, or enter the data in your computer and use a program to generate the plot. Not long ago I was occupied with this drill, and it seemed to me that there ought to be a way to automate the process. A little looking around turned up a lot of information on the subject of RF signal measurement.[1, 2, 3, 4] After several false starts I came up with a design based on the AD8307 that seemed promising.

A block diagram of my measurement system is shown in Figure 1. It consists of a computer having at least one serial port, a signal generator that can be controlled by commands sent to its serial port, the filter to be evaluated (DUT), a couple of attenuators and a bit of hardware to tie everything together: a detector module and a control module. The main component of the detector module is an AD8307, which generates a DC voltage proportional to the power of the input RF signal. The control module has two functions accomplished by a microprocessor: relaying command strings from the computer to the signal generator and acquiring and sending the power measurements to the computer.[5]

Figure 2 is a photograph of the two modules mounted in a small enclosure. Following the advice of Hayward and Larkin, I mounted the detector board in a smaller enclosure made of 24-gauge copper sheet (the top cover has been removed for

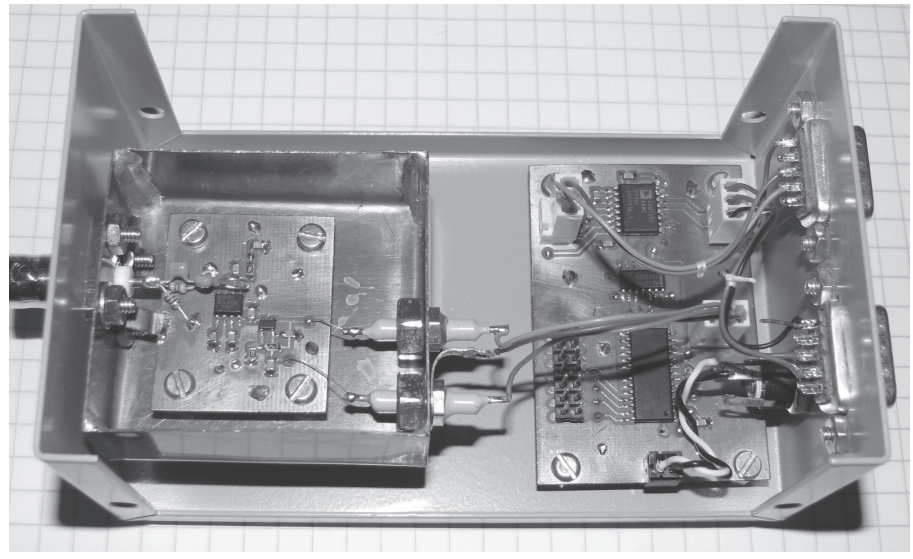


Figure 2 — The photo shows the two modules of the measurement system mounted inside a small metal enclosure. The detector board is also mounted inside a separate enclosure. For this photo, the top cover of that enclosure has been removed.

[1]Notes appear on page 6.

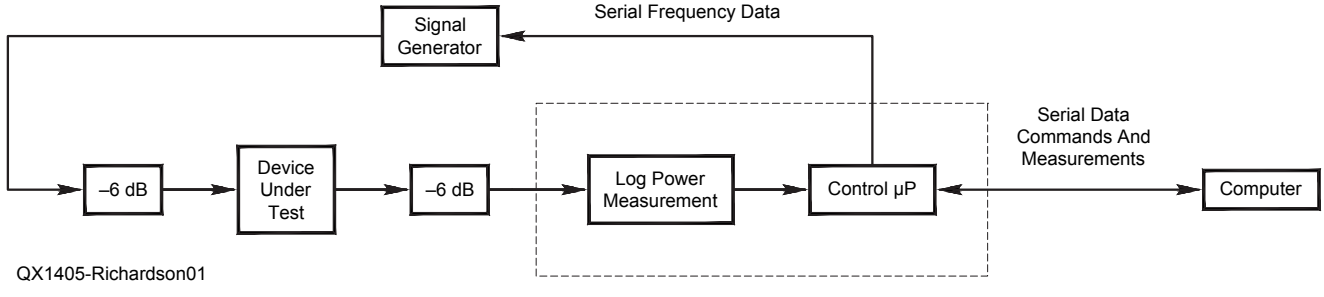


Figure 1 — This is a block diagram of the filter measurement system.

viewing in this photo). A separate enclosure for the detector board might not be necessary if the larger enclosure is RF tight.

Electronics

Figure 4 is the schematic of the detector board and is a variation of the circuit described in note 2. All of the components are mounted on the top side of a small piece of 1/32 inch double-sided circuit board. The initial version used surface-mount parts for the input network but the high-frequency response, above 100 MHz, was lower than that shown in Figure 2 of the Hayward/Larkin article and I thought small through-hole components might be an improvement. There was no significant change, however. With a 5 V V_{CC}, the output of the AD8307 ranges from about 200 mV to about 2.5 V. An op-amp with a gain of gain 1.25 increases the maximum output voltage to slightly less than the maximum ADC input of 3.3 V. I had originally planned to operate the AD8307 at 3.3 V but I found that the dynamic range was considerably reduced.

The microprocessor series I'm most familiar with is the TI MSP430, inexpensive, low power devices that have a very nice instruction set and excellent open-source development and debugging tools. I chose an MSP430F1232 for this project because it is the smallest processor in this series with an ADC (10 bits).

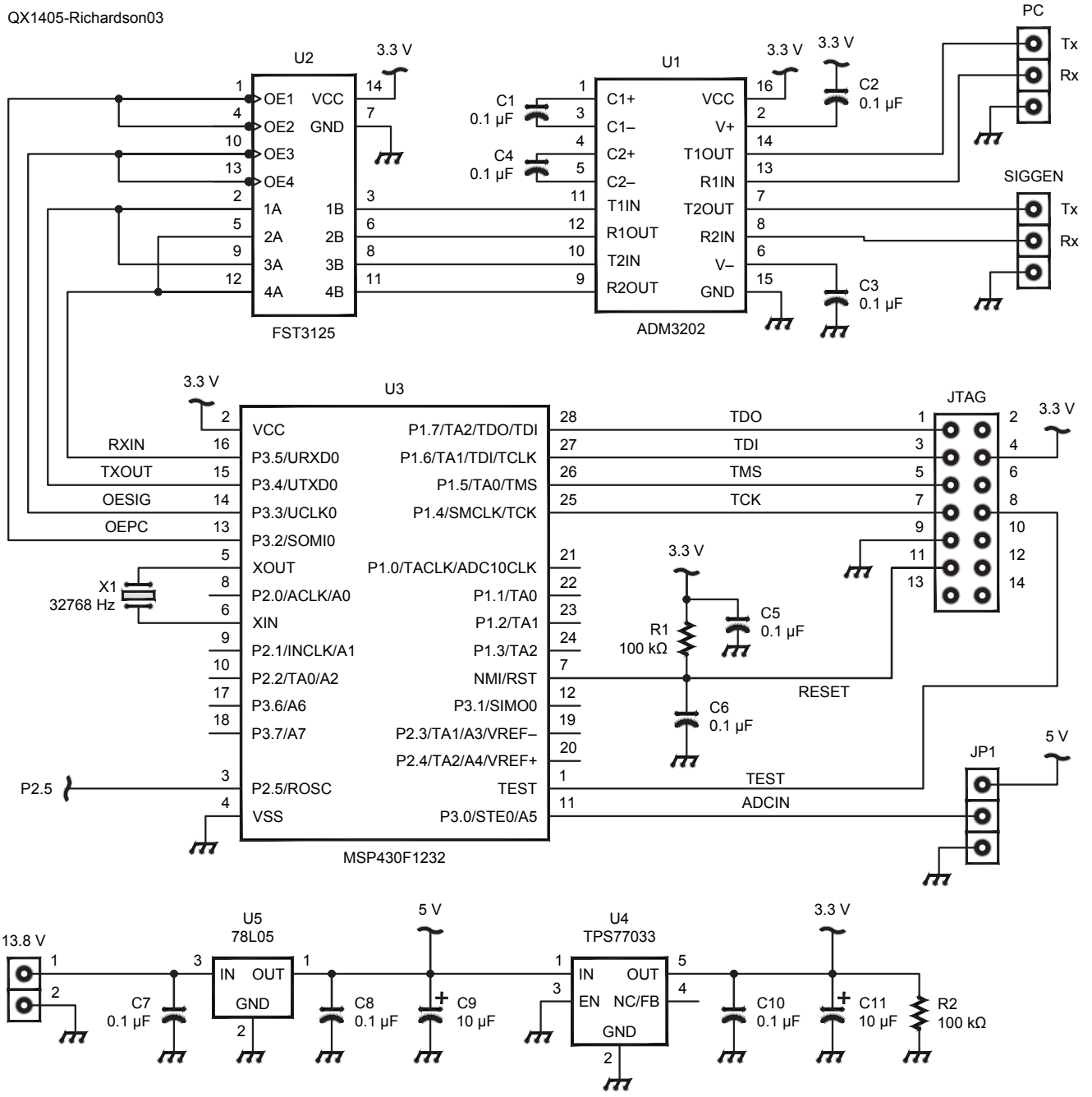


Figure 3 — Here is the schematic diagram of the microprocessor board.

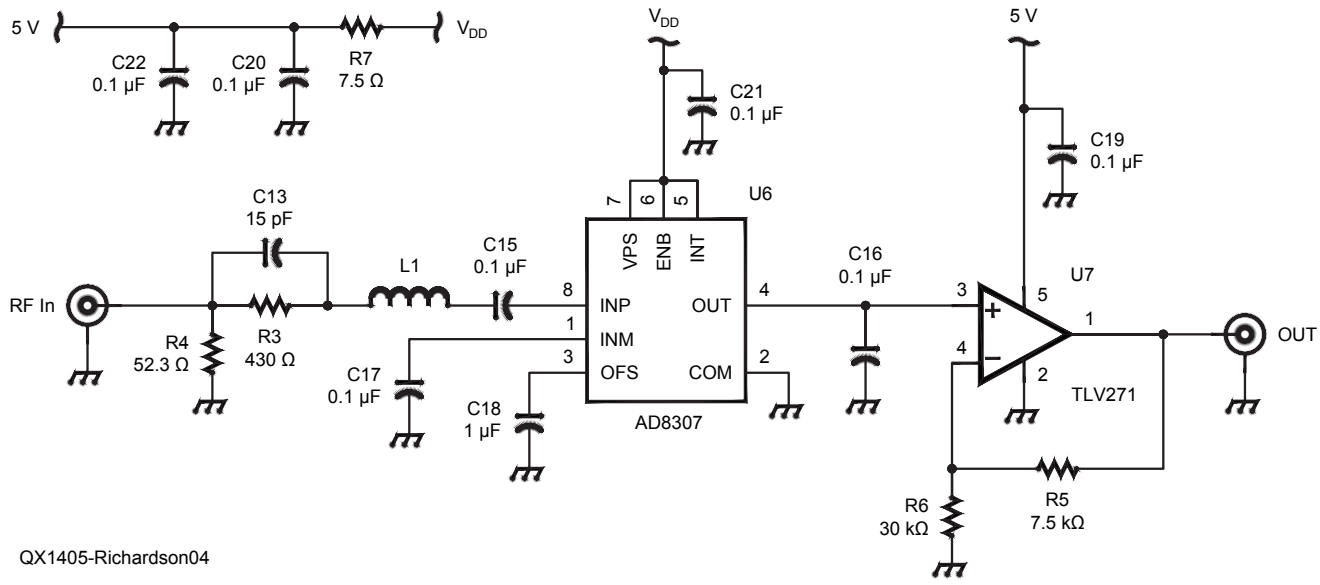


Figure 4 — This schematic diagram shows the circuit of the detector board.

Figure 3 is a schematic of the processor board. I had initially planned to include an FT245R USB/Serial IC on this board to provide a USB channel to the computer. I created a nice looking schematic but progress in routing the traces on the board quickly ground to a halt. With only two layers it was impossible (for me at least) to route the connections between the FT245R and the microprocessor. It might have been possible to connect the two devices with wires, but it would have been a mess. So I adopted a serial approach.

The ADM3202 is a dual RS-232 line driver/receiver device. The SN74CBT3125 is a quad FET switch that connects one of two serial channels to the microprocessor. This board supplies the power for both boards, hence the two voltage regulators. The total current consumption is about 23 mA.

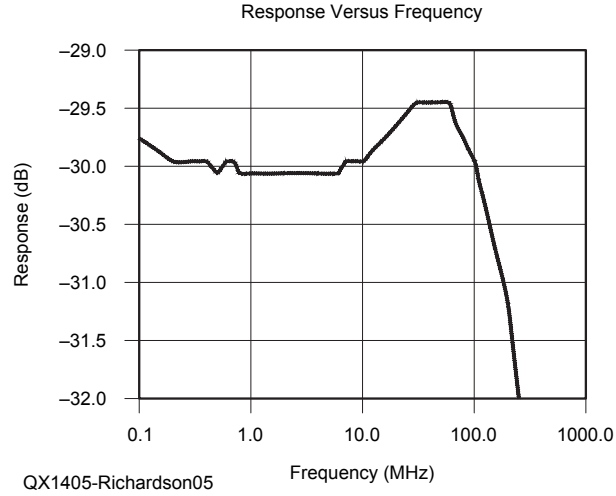


Figure 5 — This graph is the plotted frequency response of the unit. You can see that with only a few tenths of a dB variation, the response is flat from 100 kHz through 100 MHz.

Performance

Figure 5 is a plot of the frequency response of the unit for a –30 dBm input; it varies a few tenths of a dB between 100 kHz and 100 MHz, but then falls off rapidly, whereas the response of the Hayward/Larkin unit is reasonably flat out to 600 MHz. I used a BNC connector for the RF input and RG-58 coax to connect to the signal generator. I assume the bandwidth of my unit would have been greater had I used an N-type connector and high quality coax.

The top portion of Figure 6 is a plot of detector output in ADC counts versus input power for a 10 MHz input signal. A regression (least squares fit) line is drawn through the data points from –70 dBm to +15 dBm. The slope of this line is the scale factor of the measurements in terms of ADC counts per dBm. Y0 is the value on the regression line for 0 dBm. The lower part of the graph shows the difference between the data points and the corresponding values on the regression line, and provides an indication of the linearity of the AD8307.

Software

The software for the MSP430F1232 was written in *C* using the mspgcc development tools.[6] The computer software was written in *Python* and consists primarily of two modules, one dealing with the communication with the microprocessor and the signal generator and the other with various higher level functions, primarily plotting.[7] The code that generates the plot of Figure 6 also serves to calibrate the system; the gain and Y0 values are saved in a file and are used by the plotting functions. An example of the use of the plotting function is shown in the code of Figure 7. This code was used to generate the plot of Figure 5. The arguments to the plot function are obvious except perhaps for "yOffset." If this argument is set equal to the total pad attenuation (typically 12 dB) minus the signal generator power level,

the peak plot values will be near zero. The function "plotFrequencyResponse" has an additional argument not shown Figure 7, *findBandwidth*, which when set to true will cause a horizontal line to be drawn at the –3 dB level and the bandwidth of the response curve to be computed and displayed on the plot. This requires that the yOffset argument be set to –12 dB (assuming 6 dB attenuators used).

The files for this project are available for download from the ARRL *QEX* files website at **www.arrl.org/qexfiles**. Look for the file **7x14_Richardson.zip**.[8] The files include the software (*Python* and microprocessor code) and the Eagle schematic and circuit board layout files.[9]

Gary Richardson, AA7VM was first licensed as KN5WHO in 1957. Interest in amateur radio waned in subsequent years due to pressure of school and work. Gary earned an MSEE degree from Michigan State University in 1967 and spent much of his career designing software for embedded microprocessors in medical systems. He was licensed as AA7VM in 1999. You can reach Gary at PO Box 228, Marblemount, WA 98267 or aa7vm@arrl.net.

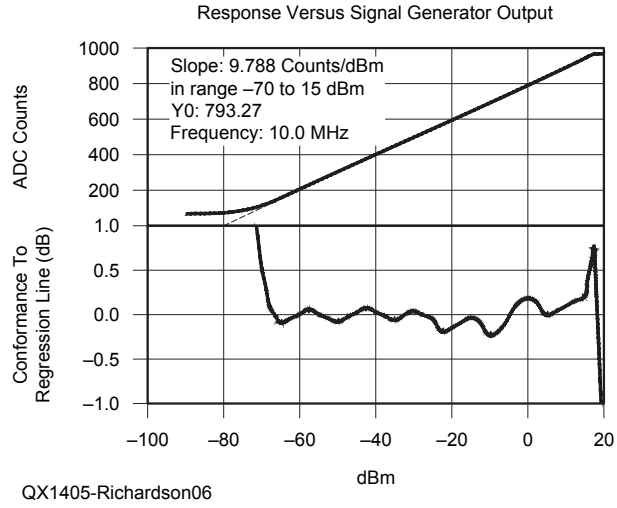


Figure 6 — The top portion of this graph shows the analog to digital converter counts versus the input power from the signal generator, which is set to 10 MHz. The lower portion of the graph shows the deviation from or the conformance of the measurements to a least squares fit of the data. You can see that the AD8307 log detector is linear from about –75 dBm through +10 or +15 dBm.

Notes

[1]Wes Hayward, W7ZOI and Bob Larkin, W7PUA, "Simple RF-Power Measurement," June 2001 *QST*, pp 38-43.

[2]Wes Hayward, W7ZOI, Rick Campbell, KK7B, and Bob Larkin, W7PUA, *Experimental Methods in RF Design*, Section 7.3. ISBN: 978-087259-923-9; ARRL Publication Order No. 9239, $49.95. ARRL publications are available from your local ARRL dealer or from the ARRL Bookstore. Telephone toll free in the US: 888-277-5289, or call 860-594-0355, fax 860-594-0303; **www.arrl.org/shop**; **pubsales@arrl.org**.

[3]Loftur Jonasson, TF3LJ/VE2LJX, "Squeeze Every Last Drop Out of the AD8307 Log Amp," May/June 2013 *QEX*, pp 29-33.

[4]For more information about the AD8307 logarithmic amplifier see the Analog Devices website: **www.analog.com/en/rfif-components/detectors/ad8307/products/product.html**

[5]If your computer has two serial ports one could be used to talk to the signal generator, eliminating one of the tasks the microprocessor would need to perform, and would simplify the hardware somewhat.

[6]For information about the mspgcc development tools see: **http://sourceforge.net/apps/mediawiki/mspgcc/index.php?title=MSPGCC_Wiki**

[7]You can find more information about *Python* at: **www.python.org/**. The plots were generated with the matplotlib package: **www.matplotlib.org/**

[8]The files for this article, including the software (*Python* and microprocessor code) and the Eagle schematic and circuit board layout files are available for download from the ARRL QEX files website at: **www.arrl.org/qexfiles**. Look for the file **7x14_Richardson.zip**.

[9]The schematic shown in Figure 4 is not exactly the same as the schematic used to generate the board because the input network components (C13, L1, R3, R4, C15) are through-hole components and are not mounted on the board, neither is the coax connector.

Figure 7 Computer Code Listing

```
from PMSA.process import PMSA_PROCESS
dirname = 'C:\Documents and Settings\Owner\My Documents\Python\PMSA'
pmsa = PMSA_PROCESS(dirname)
freqs = [0.1*k for k in range(1, 10)]
freqs += [k for k in range(1, 10)]
freqs += [10*k for k in range(1, 10)]
freqs += [100*k for k in range(1, 5)]
freqs = array(freqs)              # Frequencies have units MHz
siggen = -30                      # signal generator power level - dBm
yH = siggen + 1                   # Max Y axis limit in dB
yL = siggen - 2                   # Min Y axis limit
yOffset = 0                       # no Y-a xis offset
title = 'Response vs Frequency'
mode = 0                          # semilog plot
pmsa.plotFreqResponse(freqs, siggen, yOffset, yL, yH, title, mode)
```

Dr Sam Green, WØPCE

10951 Pem Rd, Saint Louis, MO 63146; **w0pce@arrl.net**

A Fully Automated Sweep Generator Measurement System — Take 3

This update describes enhancements achieved by substituting a daughtercard with a programmable Si570 crystal oscillator for the NJQRP DDS daughtercard and requisite changes to the addressing circuitry, drivers, and logarithmic detector.

The first two takes of my fully automated sweep measurement system were based on the American QRP Club/New Jersey QRP Club Direct Digital Synthesis (DDS) Daughtercards.[1, 2] The earlier version of the NJQRP card uses the 0 – 30 MHz Analog Devices AD9850 DDS and the current version uses the 0 – 60 MHz AD9851. Both synthesize 10 bit approximations of a sine wave with digital to analog converters to provide readily filtered sinusoidal waveforms.

This update substitutes the KangaUS/AAØZZ Si570 daughtercard to perform measurements at still higher frequencies. Both the NJQRP daughtercard and the KangaUS daughtercard have similar dimensions and pinout, as both mate and operate with the PIC-EL.[3, 4] Usage in my system requires modifications to the base circuit board and to the daughtercard.

Differences between the Analog Devices DDS and the Silicon Labs Si570 are many. One major difference is that the Si570 provides logic level outputs rather than synthesized sinusoidal waves. Another major difference is that we control the Si570 through an inter-integrated circuit (I^2C) interface rather than the serial (or parallel) load interface of the AD985x.[5]

[1]Notes appear on page 15.

Figure 1 shows that there are twenty different standard versions of the Si570 in the data sheet, and Silicon Labs is happy to sell you additional custom versions.[6] The standard twenty offer four different logic families, three power supply ranges, and a choice of a high or low signal on an output enable pin. Within these twenty, there are three options for temperature stability and four possible frequency ranges. Whew! The Si571 variant also offers voltage variable fine tuning.

The KangaUS Si570 daughtercard is the same one that Craig Johnson, AAØZZ, shows in his *QEX* article of July/August 2011.[7] The particular Si570 version that KangaUS provides is the lowest cost 3.3 V CMOS "C" version with Output Enable high and the lowest frequency range of 10 – 160 MHz. I bought one of these from KangaUS to perform development, plus a bare board to mount the fastest Si570 "A" version of the IC, which I obtained from my friend Herb Ullmann, AF4JF. The "A" version operates from 10 - 1417.5 MHz with two gaps in the operating range, so I can sweep it from 10 – 945 MHz, from 970 – 1134 MHz, and from 1213 – 1417.5 MHz. The much less expensive "B" version operates from 10 – 810 MHz, and is a "best buy" unless you really need the highest frequency regions.

Versions with the best temperature stability of 7 parts per million use a different set of programming registers than the less stable versions, so avoid them unless you are prepared to recompile the software.

Note that the Si570 only operates down to a minimum of 10 MHz, so you still need the NJQRP DDS to operate at lower frequencies.

I changed the circuitry and the printed circuit card to accommodate the requirements of the I^2C interface but kept both compatible with the NJQRP AD9850 and AD9851 daughtercards.

The I^2C hardware interface and software come from Maxim, but I modified the interface to accommodate the non-inverting open collector 74LS07 rather than the inverting 74LS05 that Maxim uses, and I modified the software to access different pins on the parallel port in order to maintain compatibility with the NJQRP daughtercards.[8, 9]

Again, this simple software doesn't work with newer versions of *Windows* unless you use a third party application to enable program control of the I/O hardware. I use *UserPort* and some folks recommend an alternative (*Direct I/O*) that I haven't tried.[10, 11]

High frequency pickup problems that I solved in the previous article returned with a vengeance. I discuss problems and solutions later in this article.

Circuit Revisions and Corrections

Take 2 of the project used six inverters to buffer the three lines from the parallel

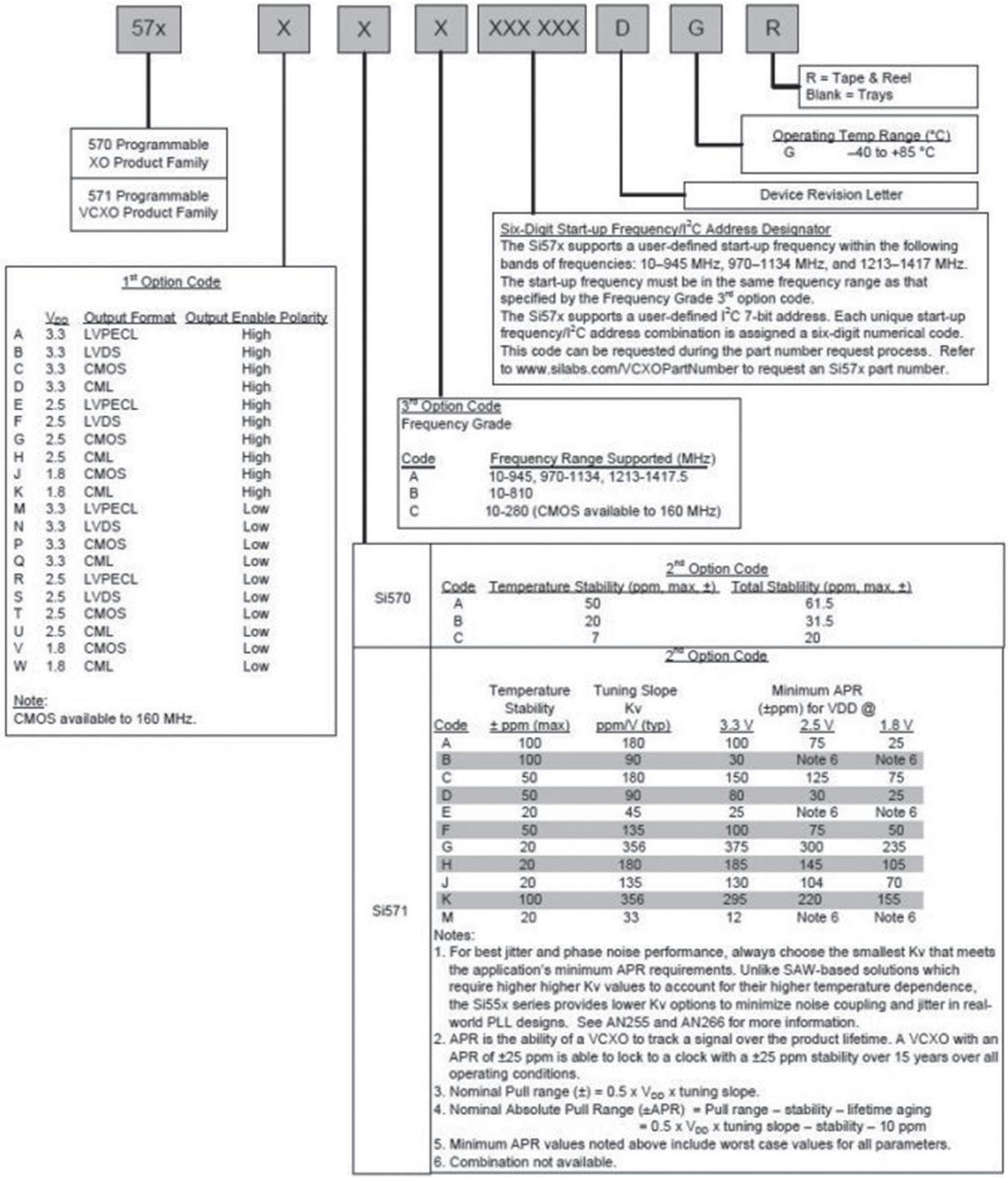


Figure 1 — Standard versions of Silicon Labs Si570.

port to the DDS daughtercard with power from the daughtercard so that an AD9851 won't receive any power when the DDS is off. The AD9851 does not power-on-reset when parallel port lines supply even limited power. Pin 4 of the DDS daughtercard provides 5 V dc to power these buffers so that all DDS power disappears with power off. The KangaUS Si570 daughtercard does not use pin 4, so I added a jumper from pin 4 to the +5 V dc regulator to power the address buffers. Figure 2 shows the daughtercard modifications. Omit C8 and convey the output directly via coaxial cable to decrease EMI.

The I²C interface requires more than three gates, and they require open collectors and pull-up resistors. Maxim designed their I²C interface and software around the open-collector inverting 74LS05. In order to use the six or fewer gates available in a single DIP package and work with both the DDS and the Si570, I could no longer waste two inverters to form a non-inverting buffer for each of three DDS lines.

Given these restrictions, I changed the Maxim software to use the non-inverting open-collector 74LS07. To maintain the ability to drive the DDS with the same interface, I further changed the Maxim software to drive different parallel port pins. No, none of this worked the first time, nor the second!

Figure 3 shows the revised schematic. Note the changes to the lines between the DB-25M connector and the daughtercard socket. Especially note the addition of the reverse signal path from pin 2 of the daughtercard socket through a non-inverting gate and back to pin 12 of the DB-25M connector. This pin is bit 5 of the status port at base address plus 1, and serves as an input for the I²C receiver.

Si570 Output Configurations

The Si570 is available with LVPECL, CMOS, LVDS, and CML outputs. The lowest-cost version of the Si570 has a single-ended CMOS output that provides several volt output signals rather than the complementary fractional volt output signals that the others provide.

The CMOS version provides sufficient output swing to overload the combination of logarithmic detector and analog-to-digital converter. I use a 6 dB 50 Ω BNC attenuator on the output to handle the excess power. I suggest adding a surface mount 6 dB attenuator chip to the daughtercard with the CMOS version. Several are available inexpensively from Digi-Key including the PAT05S6CT-ND and the PAT126CT-ND. Mini-Circuits offers the LAT-6+, but their minimum order quantity is 20, and I found

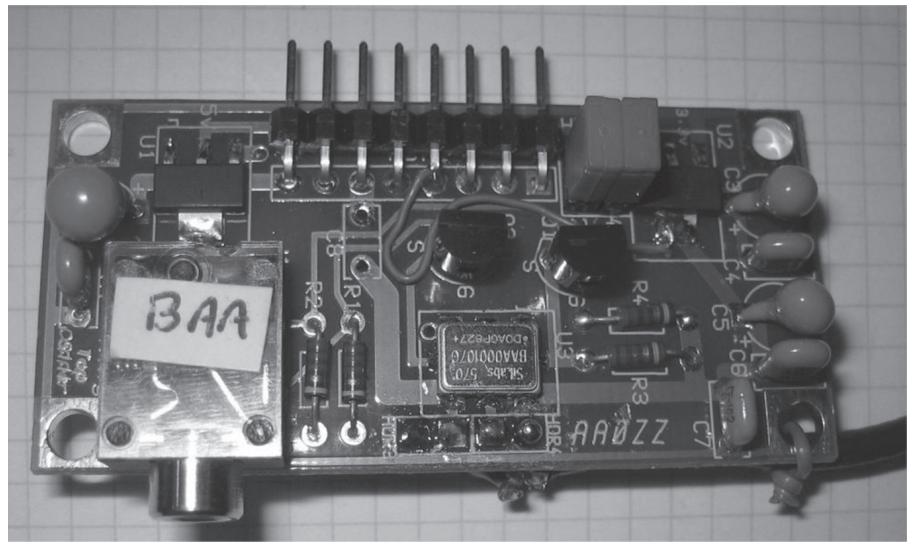


Figure 2 — AAØZZ KangaUS Si570 daughtercard – Note the jumper to bring +5 V to pin 4, and the omission of C8.

Table 1
Parts List

Capacitors

C1	22 pF
C2, C3	1 μF or 100 μF for low frequency operation
C4	1 μF
C5	0.001 μF
C6, C7	0.100 μF

Resistors

R1, R2, R3, R15	4.7 kΩ
R4	10 kΩ
R5	20 kΩ
R6, R7	40 kΩ
R8, R10, R11	3.9 kΩ
R9	1 MΩ
R12	51 Ω
R13	22 kΩ
R14	100 kΩ

Voltage Reference Diodes

D1	LT1004-2.5 in TO-92
D2, D3	LT1004-1.25 in TO-92

Active Components

74LS07 or 7407 or 74C07
MX7543 DAC
MAX110 ADC
LM324 Quad Operational Amplifier
AD8307 and/or ADL5513 logarithmic detectors

NJQRP DDS or AA0ZZ KangaUS Si570 daughtercard

ADL5513 Components in Figure 3 are all surface mount except L2 uses a six-hole ferrite core in **Figure 4** and ferrite beads in **Figure 8**.

Standard ham tolerance of half to double is acceptable for all components except R12, which should be near 50 Ω, and R4, R5, R6, and R7, which are not critical in value but should be very close to a ratio of 1:2:4:4.

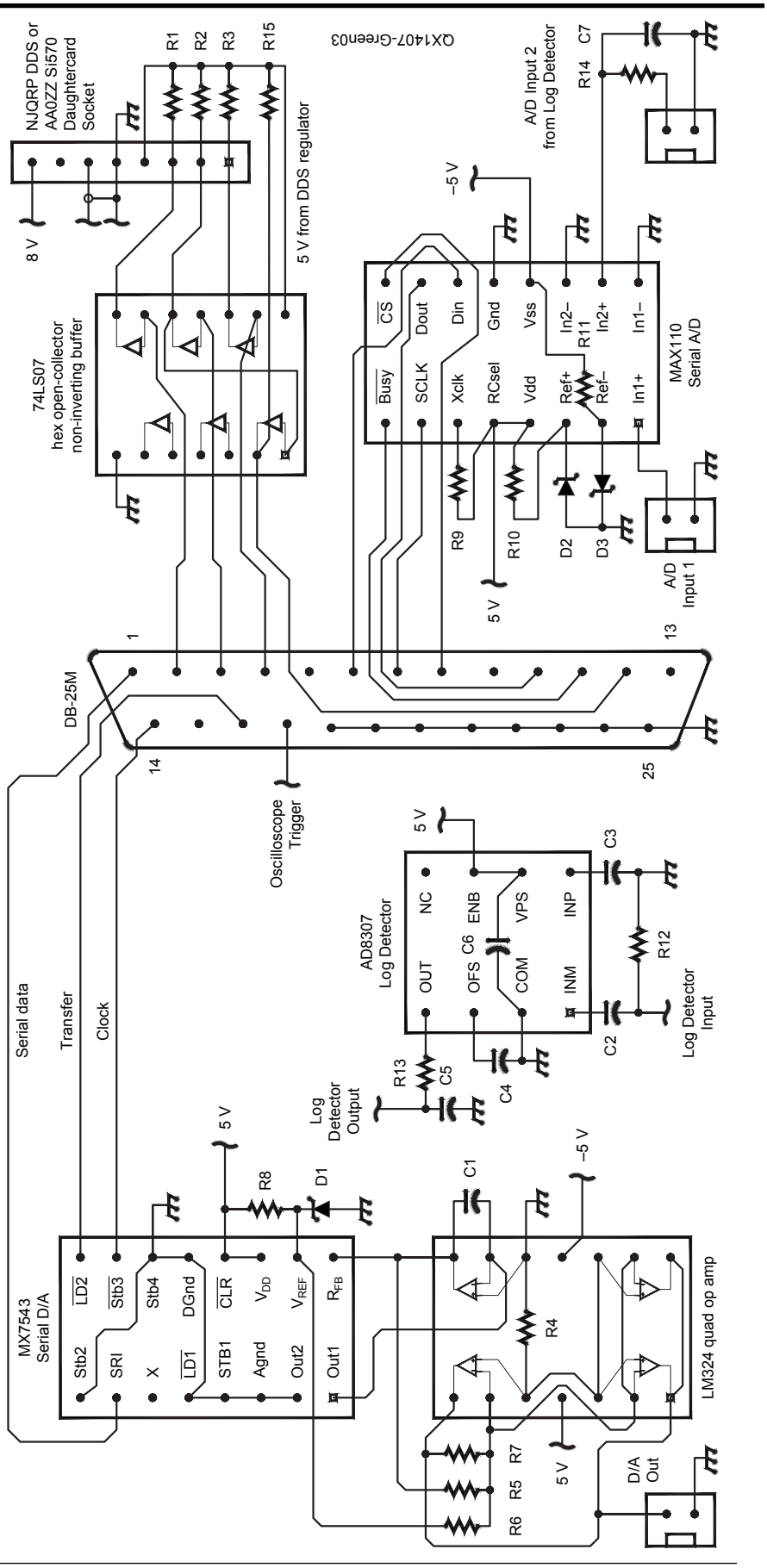


Figure 3 — Pictorial schematic with AD8307 logarithmic detector for sources below 500 MHz shows non-inverting open-collector buffers to daughtercard.

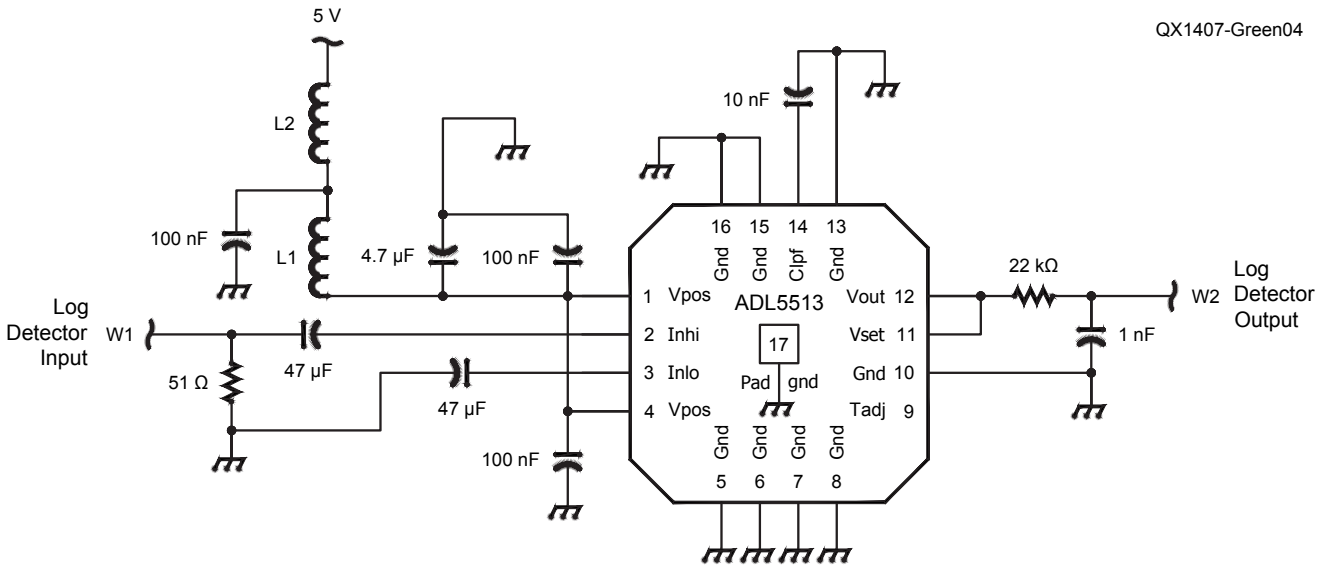


Figure 4 — ADL5513 logarithmic detector schematic diagram.

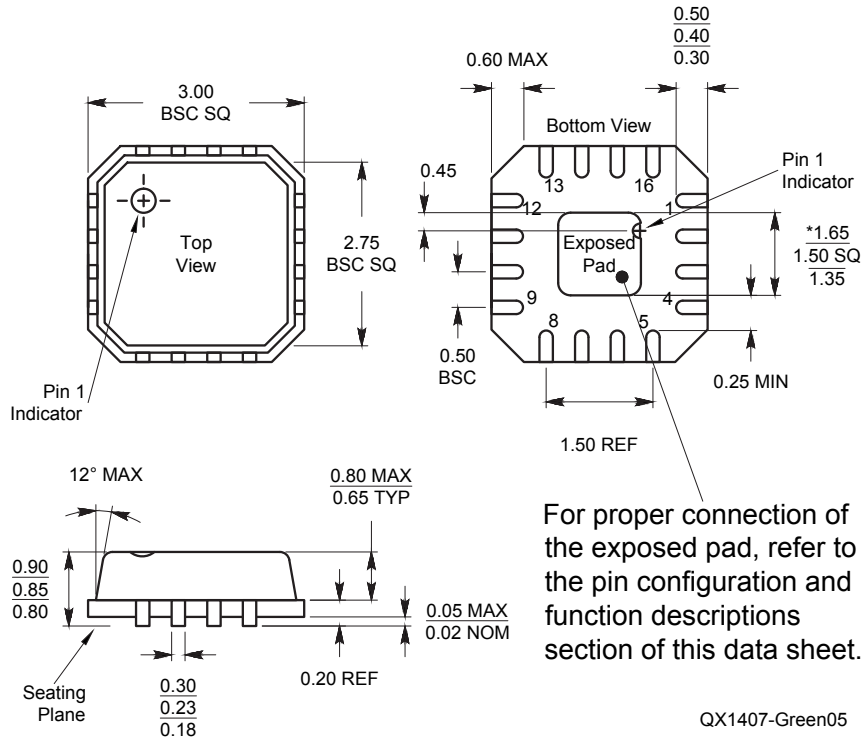


Figure 5 — Lead frame chip scale package [LFCSP_VQ] (3 mm × 3 mm).

them very uncooperative with requests for information in the past.

My Si570 has complementary LVDS outputs that expect a 100 Ω differential load. Instead, I placed a 50 Ω resistor between one output and some RG174/U coaxial cable to the front panel connector. I placed a 100 Ω resistor between the complementary output and ground. The reverse termination minimizes reflections from poorly matched loads.

Frequency Range Issues

Silicon Labs specifies the CMOS part from 10 to 160 MHz. The one I evaluated works fine to 170 MHz. The AD8307 logarithmic detector in the previous articles works to 500 MHz and need not be changed for the low-frequency CMOS version. The AD8307 is not suitable for use with any of the high frequency versions of the Si570. I experimented with the 2.5 GHz AD8313 and the 4 GHz ADL5513 logarithmic detectors and chose the latter because it has a wider dynamic range. Figure 4 shows the schematic for the ADL5513.

Figure 5, from the ADL5513 data sheet shows that the chip is very difficult to work with because the 16-Lead Lead Frame Chip Scale Package [LFCSP_VQ] has contact pads underneath instead of leads.

I laid out a printed circuit board to accommodate both the ADL5513 and the AD8307, because I wasn't sure I could make the chip scale part work. Figure 6 shows an early photo of the new board before I learned to align the chip scale package properly. Figure 7 shows the detail of poorly and properly aligned chips. Both actually worked.

Attachment of such parts requires solder paste and a hot air rework station for assembly and the many repairs. I purchased my own after borrowing one several times from my friend Lee Johnson, NØVI.

Figure 8 shows significant pickup from the CMOS Si570, but this would be greatly reduced with a surface mount 6 dB pad on the daughtercard and coaxial cable to the front panel instead of using pin 6 for the RF output path as with the DDS daughtercards. Remove or do not initially install C8 on the Si570 daughtercard so that RF does not reach pin 6.

Pickup from the LVDS Si570 to the AD8307 is low because the level is lower than with the CMOS Si570, because all RF from the daughtercard passes through the coaxial cable.

Early attempts at coupling high frequency output signals from the LVDS Si570 led to a lot of pickup between the Si570 and the ADL5513 logarithmic detector on the board. Shielding and common mode filtering did

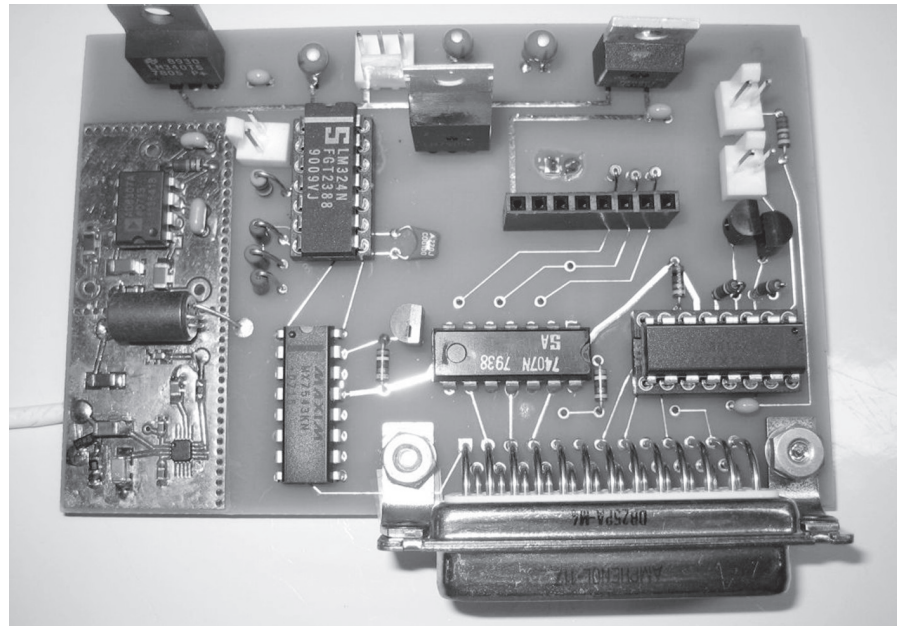


Figure 6 — Photo of the new board with both AD8307 and ADL5513 logarithmic detector chip. High Frequency Si570 versions use ADL5513. CMOS versions use original AD8307. Both are populated in this photo, but only the ADL5513 receives power and an input.

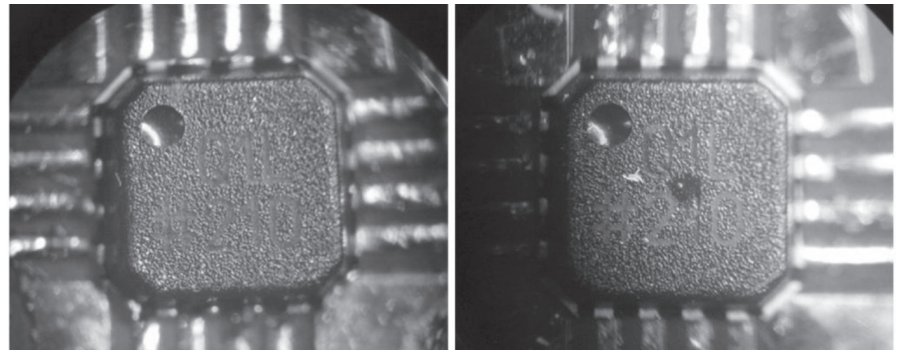


Figure 7 — Misaligned and aligned ADL5513 chips prove difficult to solder.

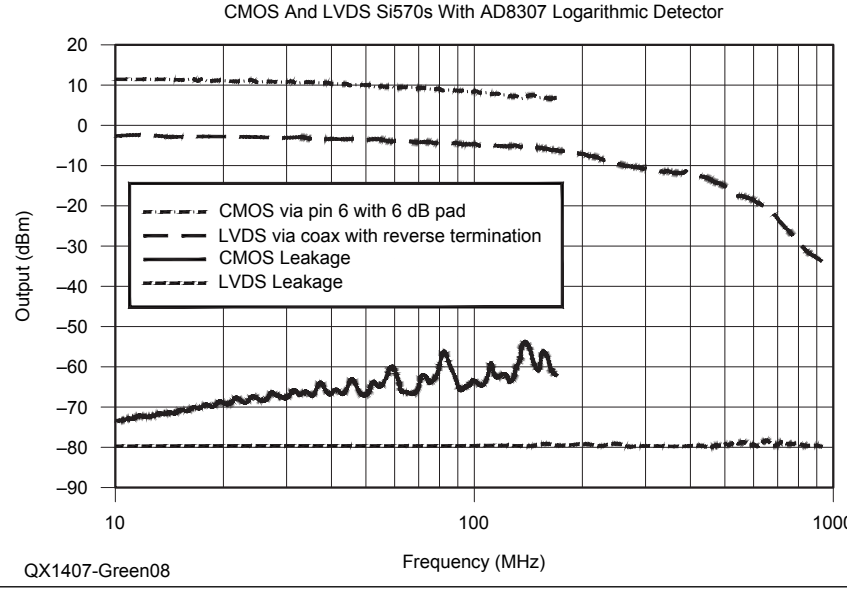
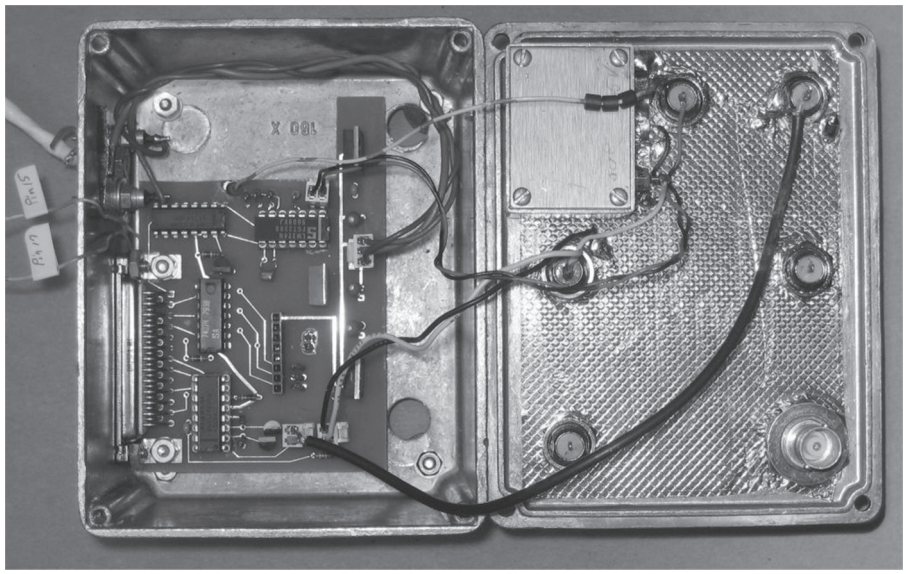
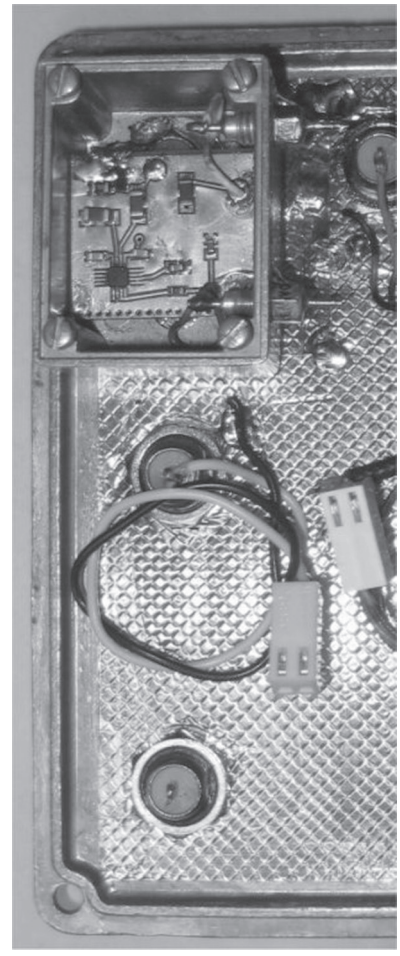


Figure 8 — CMOS and LVDS Si570s drive a 500 MHz AD8307 logarithmic detector.



(A)



(B)

Figure 9 — Part A shows the logarithmic detectors removed from the main board and the ADL5513 placed in the Pomona Box with the cover on. Part B shows the ADL5513 logarithmic detector in the Pomona Box, without the cover.

not help. I removed the ADL5513 portion of the circuit from the board and mounted it within a Pomona Model 3754 shielded case, the smallest I could find.[12] The completely shielded configuration in Figures 9A and 9B solves all pickup problems.

Alternatively, the output configuration with coaxial cable directly to the daughtercard may adequately reduce the pickup, but I already removed the ADL5513 logarithmic detectors from all available boards and can no longer evaluate this configuration. Figure 10 shows that I bring one LVDS output to the front panel via coaxial cable, with a reverse termination and terminate the complementary LVDS output for balance.

A disappointing feature of the Si570 also pointed out by AAØZZ is that the output ceases between large frequency changes. In my drivers, all frequency changes are large, so there is a 240 µs dropout preceding each measurement. This doesn't matter for the swept measurement programs where the frequency steps to the next value and an analog-to-digital converter measures the output of the logarithmic detector for some period. It does matter in the continuously swept programs for oscilloscope display. In these, a glitch appears at each start and stop because I couple the output through a capacitor to remove the dc offset of the output. For this reason, I select a small value of 300 – 1000 pF for the capacitor into the coaxial cable in this figure. The resulting narrow pulses are now barely noticeable.

Note that I like to use copper tape with conductive adhesive for good grounding when I work with other materials.

New DDS Software Drivers for previous versions

With multiple instances of this equipment operating with two versions of the NJQRP DDS daughtercard and now two versions of Si570 and slightly different values of calibration parameters, configuration control became a problem. Each unit was slightly different, even if it was just the offsets or the values of the reference diodes for the digital to analog and analog to digital converters. Each driver for each version had these constants embedded, and I recompiled each driver for every version. This was getting old.

I revised all of the earlier drivers to accept external calibration files, so that a simple text file with appropriate calibration parameters accompanies a particular piece of hardware. Each program now reads this calibration file on startup and uses the appropriate values in its operation, so the same program now runs similar kinds of hardware and accounts for inherent and minor differences.

The DDS calibration file format is:

DACVref DACoffset ADCVrefPos ADCVrefNeg ADCoffset LogDetSlope LogDetIntercept RefClock Base

A corresponding DDS-60 calibration file looks like this:

2.484 0.0017 1.232 1.232 +0.0005 0.025 –86.0 180000000 0x378

With this system in place, I no longer modify and recompile the drivers to accommodate calibration differences.

Further, I use the reference clock to discriminate between the NJQRP DDS-30 and the NJQRP DDS-60. If the reference clock is around 100 MHz, the programs run an AD9550 direct digital synthesis chip, and if the reference clock is around 180 MHz, the programs run an AD9551 direct digital synthesis chip. Small changes to these numbers further allow correction to the operating frequency, if you have a better frequency measurement setup than I do. Finally, if the reference clock is 10 MHz, the DDS programs complain that the Si570 is present instead of a DDS and tells me to use a different program.

With this structure in place, I then wrote self-calibration software. I calibrate the ADC from a reference source that I measure with my best digital multimeter, a Fluke 77. The program derives the three ADC parameters and writes them to the calibration files. Another program then calibrates the DAC against the ADC and writes its calibration parameters to the calibration file. With these two functions calibrated and in agreement,

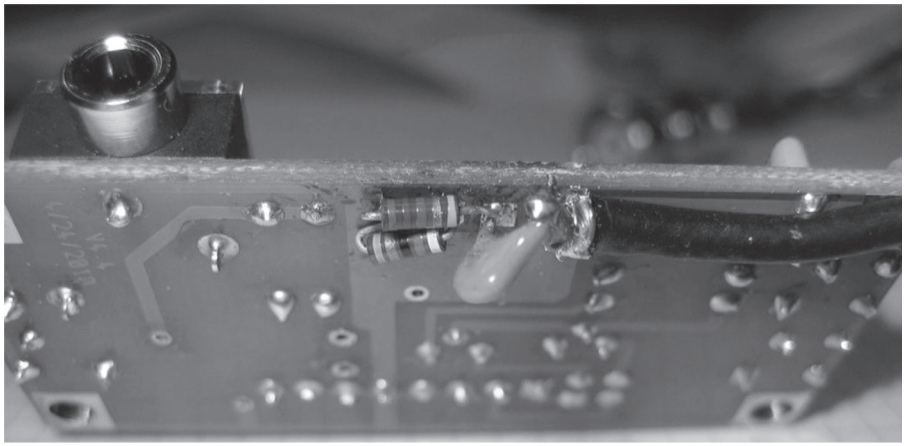


Figure 10 — Coaxial cable connects directly to the Si570 daughtercard.

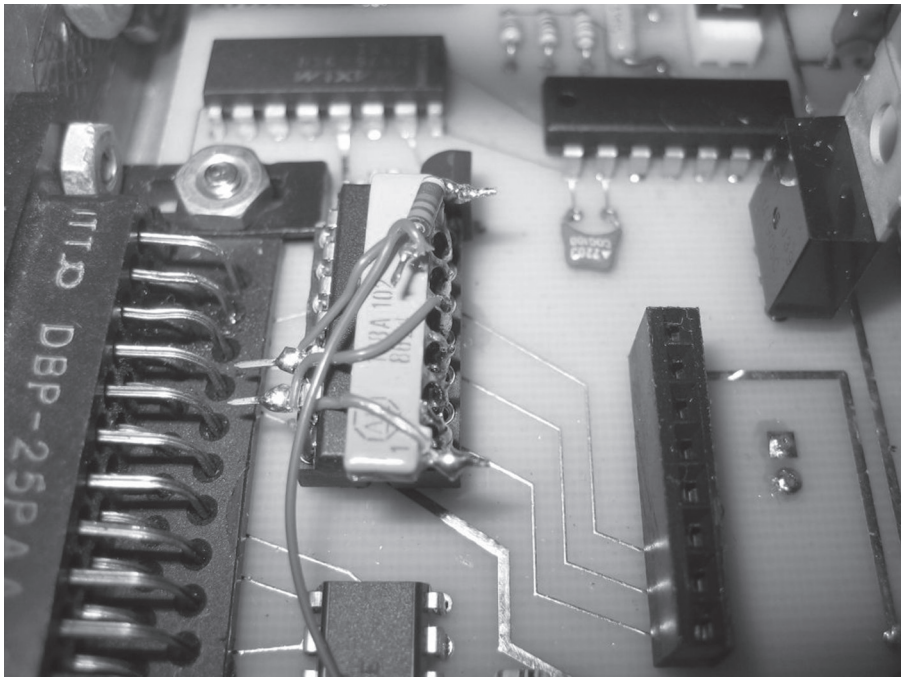


Figure 11 — Here is the Take 2 circuit board with the updated open-collector interface.

the Transfer.exe measurement becomes precise to within a millivolt.

I measure the response of the logarithmic detector and use a spreadsheet to derive the slope and intercept so that RF power measurements become fairly accurate, at least at the frequency I measure. I enter the slope and intercept into the calibration file manually with a text editor. I no longer adjust two trim-pots in a sometimes vain attempt to make all logarithmic detectors behave the same. Instead I just measure their performance without adjustments and enter the parameters into the calibration file.

I similarly enter the reference clock frequency and the parallel port base address into the calibration file manually.

While implementing the modifications to accommodate the calibration files, I found several errors in the repetitive scanning DDS drivers that no one has yet complained about. These are fixed, so you can upgrade from the programs that originally accompanied the two previous papers. I also added a little functionality to the repetitively swept DDS programs to increase or decrease delay between scans with two more keys on the numeric keypad.

Si570 Software Drivers

I revised all previous programs and then adapted them to run the Si570 daughtercard in the same manner as they ran the NJQRP DDS daughtercards, except that I added an upper frequency limit called FreqLimit to the Si570 calibration file, because each of the many Si570 versions has a different upper frequency capability. Annoying things happen when a program sweeps the Si570 much above its upper frequency limit, ultimately requiring a power-on reset.

No Kits Available

I have not produced any kits for this project at the time of this writing. I use ExpressPCB, which is not a low cost solution to producing a large number of boards.[13]

A kit that would use the 160 MHz CMOS Si570 may leave the ADL5513 circuitry unpopulated but still requires installation of several larger size surface mount chips for the AD8307 logarithmic detector circuitry. Larger chips are easier to install one at a time than smaller chips. I have not tried to install several at one time in some kind of oven.

A kit that would use the high frequency Si570 variants requires the ADL5513 Lead Frame Chip Scale Package [LFCSP_VQ]. That chip requires special tooling, so the whole logarithmic detector portion of the board would have to be pre-assembled.

Conclusion

The AAØZZ KangaUS Si570 daughtercard signal source greatly expands the spectral range over which my swept measurements system is useful. I am so grateful that *QEX* brought this product to my attention that I finally subscribed to this fine journal. I thank friends Herb Ullmann, AF4JF, and Lee Johnson, NØVI, for their contributions. I thank Analog Devices, Linear Technology, and Maxim for the sample devices they provided.

Remember that you still need the NJQRP DDS daughtercard to operate below 10 MHz. Yes, I considered putting both an AD985X DDS and an Si590 programable oscillator in the same instrument but haven't yet tried to make that work.

I think about converting to a USB interface all the time. USB does not allow the rapid bit twiddling that I require, so I would have to add local intelligence just as Dr. Thomas Baier, DG8SAQ, did in his *QEX* article.[14]

And finally, yes, I was able to modify the circuit of "Take 2" to accommodate the new open-collector gate circuitry so that it successfully hosts the Si570 daughtercard. **Figure 11** shows the result.

Software Download

All programs are available for download from the ARRL *QEX* files website.[15] Description of the software operation accompany the previous "Take 2" article. The new versions use calibration files, and

there are programs to enable semiautomatic calibration. The GNU Public License Statement is included in each program. That statement says:

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.[16]

Dr Sam Green, W0PCE, is a retired aerospace engineer. Sam lives in Saint Louis, Missouri. He holds degrees in Electronic Engineering from Northwestern University and the University of Illinois at Urbana. Sam specialized in free space and fiber optical data communications and photonics. Sam became KN9KEQ and K9KEQ in 1957, while a high school freshman in Skokie, Illinois, where he was a Skokie Six Meter Indian. Sam held a Technician class license for 36 years before finally upgrading to Amateur Extra Class in 1993. He is a member of ARRL, a member of the Boeing Employees Amateur Radio Society (BEARS), a member of the Saint Louis QRP Society, and breakfasts with the Saint Louis Area Microwave Society. Sam is a Registered Professional Engineer in Missouri and a life senior member of IEEE. Sam has authored 17 patents, and has one more patent application pending.

Notes

[1]Dr Sam Green, W0PCE, "Fully Automated DDS Sweep Generator Measurement System," *QEX*, Nov/Dec 2008, pp 13-32.

[2]Dr Sam Green, W0PCE, "Fully Automated DDS Sweep Generator Measurement System – Take 2," *QEX*, Sept/Oct 2012, pp 14-24.

[3]Order the Kangaus Picel-iii kit at: **www.kangaus.com/content/picel-iii**

[4]Details about the American QRP Club PIC-EL 160 project board are available at: **www.amqrp.org/elmer160/board/index.html**.

[5]See **en.wikipedia.org/wiki/I2C**.

[6]Download the Silicon Labs (Silabs) Si750 data sheet at: **www.silabs.com/Support%20Documents/TechnicalDocs/si570.pdf**

[7]Craig Johnson, AA0ZZ, "Programmable PLL (Si570) Local Oscillator for HF Receivers, Transmitters and Transceivers," *QEX*, Jul/Aug 2011, pp 3-16.

[8]For a good discussion of how to use the Maxim I²C interface with a computer parallel port, go to: **www.maximintegrated.com/app-notes/index.mvp/id/3230**

[9]For a description of the *C* source code for the Maxim I²C interface, go to: **http://pdfserv.maximintegrated.com/en/an/AN3315.pdf**

[10]For information about the UserPort interface, used to control I/O hardware in Windows, see: **http://hem.passagen.se/tomasf/UserPort/**

[11]**http://pdfserv.maximintegrated.com/en/an/AN3317.pdf**

[12]I used a Pomona Model 3754 shielded case to shield the ADL5513 logarithmic detector IC from the rest of the circuit. See **www.pomonaelectronics.com/pdf/d3754_1_01.pdf**

[13]For small quantities of circuit boards, I use Express PCB. See their website for details: **www.expresspcb.com**

[14]Dr Thomas C. Baier, DG8SAQ, "A Low-Cost, Flexible USB Interface," *QEX*, Jan/Feb 2008, pp 11-15.

[15]All of the files associated with this article are available for download from the ARRL *QEX* files website. Go to **www.arrl.org/qexfiles** and look for the file **7x14_Green.zip**.

[16]You can read the full text of the GNU Public License at: **www.gnu.org/licenses/**

Thomas M. Alldread, VA7TA

7056 Railway Ave, Courtenay, BC V9J 1N4, Canada; **VA7TA@telus.net**

Android Wireless Project Control Part 2 — Example Application: NimbleSig III RF Sweep Generator Wireless Tablet Controller

This installment describes the Android tablet controller application, focusing on the graphical use interface (GUI) features.

Part 2 of this series will provide an overview description of the NimbleSig sweep generator tablet controller application, which forms the basis for the series. This application has proven to be a rewarding project for the author's first attempt at Android software development. It was chosen mainly because the benefits describe in Part 1 of the article are particularly appealing for this type of application. The thought of the ability to easily document and share swept frequency test results was particularly appealing. As the NimbleSig III module was developed previously, I had freedom to focus my attention on breaching the new frontiers of Bluetooth and Android.[1] A controller application complex enough to use many of the features offered by the Android operating system was desired and this application fulfilled that wish. I think it should be an interesting reference project for those who wish to make something similar.

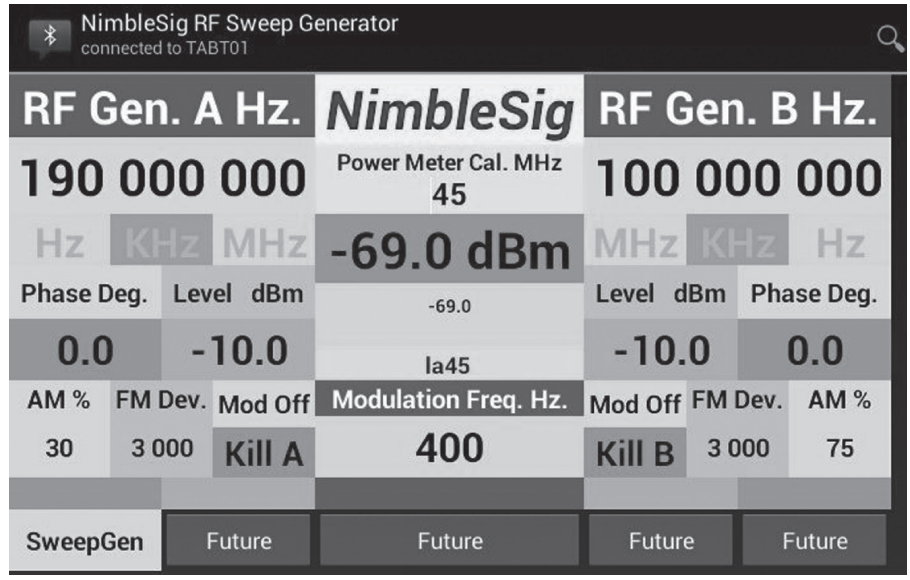


Figure 1 — Main NimbleSig controller graphic user interface.

The Main NimbleSig GUI Control Panel

Figure 1 is a screen capture of the main GUI control panel. It permits the setting of the two RF carrier frequencies, phase and level along with the modulation type, modulation index, modulation frequency for each of the two RF outputs from my NimbleSig generator. It also provides power meter calibration control, an RF power measurement display and a button for invoking the sweep generator function.

NimbleSig Controller Application Components

Figure 2 is a block diagram of the various source files that comprise this controller app. The upper half of Figure 2 shows the various *Java* source code files used to define the 12 activities used by the app. Although the large number of source code files may seem like a lot of complexity that would involve a lot of time to write, many of the activities associated with the buttons are very similar. Thus, once the first activity was defined and

[1]Notes appear on page 22.

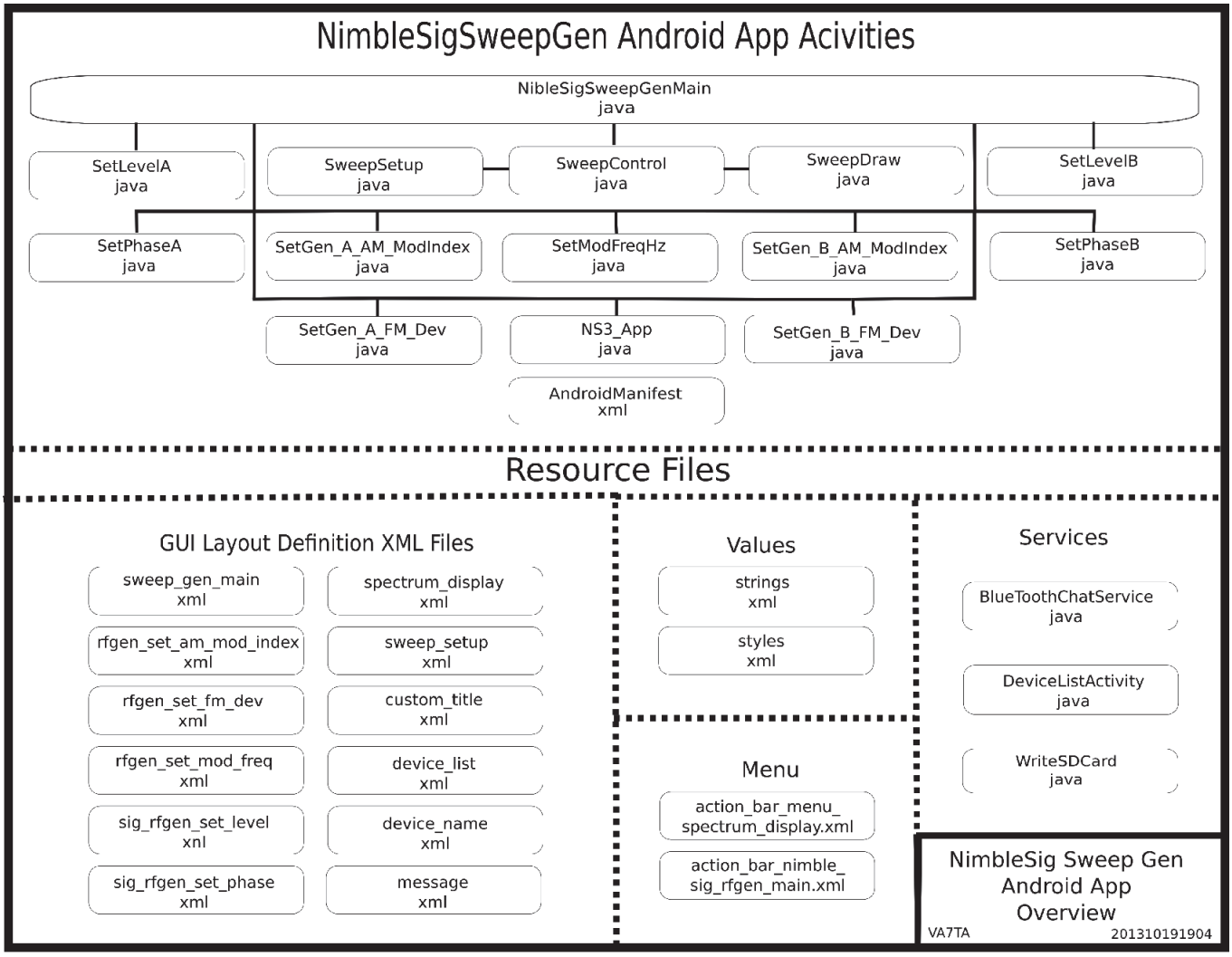


Figure 2 — NimbleSig sweep generator android application block diagram.

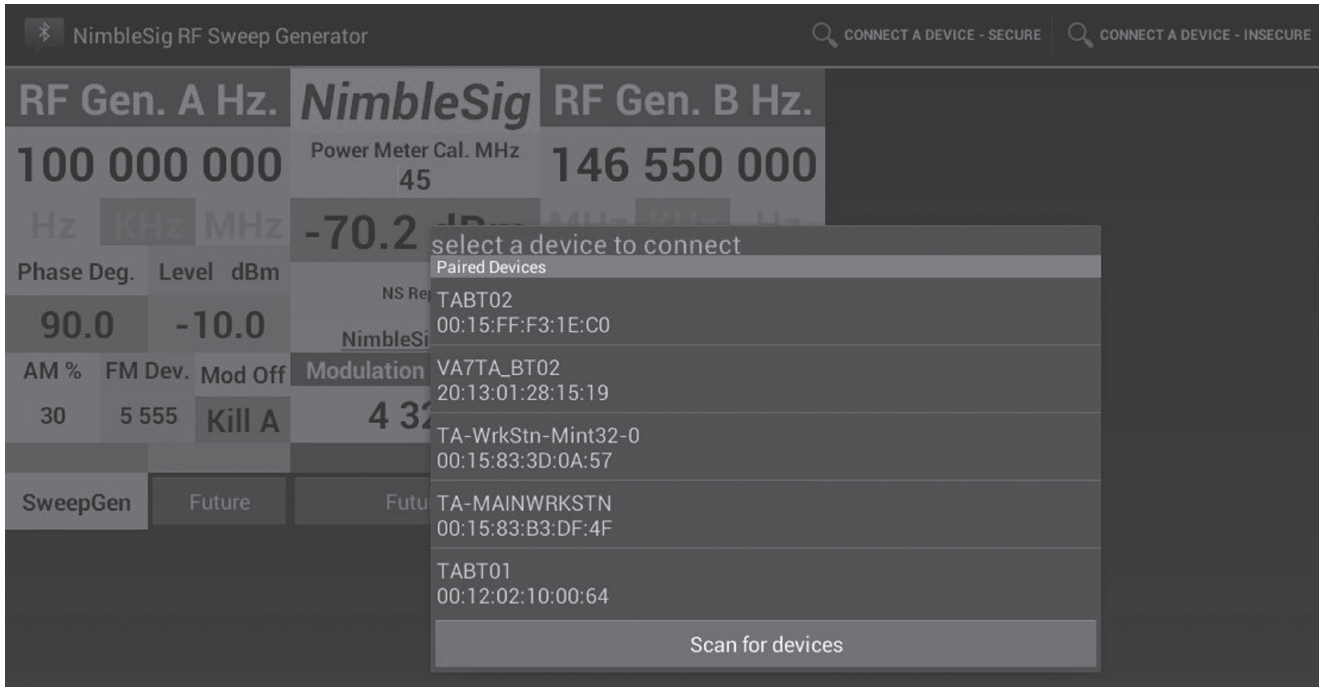


Figure 3 — Bluetooth connect dialogue.

tested, many of the same logic statements were used to form *JAVA* statements for the other buttons. In many cases only the variable names needed to be changed to adapt the same code for reuse. Thus the time required to write code for the activities needed to support additional buttons was significantly reduced compared to the initial endeavors where all the logic needed to be thought out and tested.

The *NimbleSigSweepGenMain.java* file, as the name implies, defines the main activity for the app. It supports the main GUI screen shown in Figure 1. The buttons within this GUI launch sub activities, which carry out the various control functions such as setting levels and frequencies. The *SweepGen* button launches the *SweepControl* sub activity, which in turn launches sub activities for setting up the sweep parameters and graphically displaying the swept frequency response results.

The *NS3_App.java* application file provides a central storage container for application status variables that need to be shared by multiple activities.

The *AndroidManifest.xml* file, which is external to the program code execution flow, is shown at the bottom center. All the app components and services must be registered within the manifest file. Additional app specifications, such as the previously mentioned styles, are also declared in the manifest file.[2]

The app resource files are shown in the lower half of Figure 2. The XML files in the lower left section of Figure 2 are layout definitions files, which are stored in the *res/layout* part of the app source code directory tree. These files contain the definitions for the various screens used by the app activities.

The *values* section, shown to the right of the layout, contains files that store constant data for the app. The *strings.xml* file contains text definitions for the various alpha/numeric constants used in the app. Rather than having the actual text character strings defined within the source code, Android supports the assignment of names to string data that refer to the associated text data declared within the *string.xml* resource file. This approach eases the adaptation of apps to the international market place as support for multiple languages can be implemented by just changing the resource file text string data. This simplifies adopting apps to new languages as the program code files do not need to be modified. This feature, which I think is a great idea for commercial applications, is optional. For this application, in the interest of simplification and to save some time, I chose to define the majority of text strings within the source code. The definitions could be shifted into resource files in the future without too much effort, should the need arise.

As the file name implies, the *styles.xml* file is where the app style is declared.

The files shown in the menu section are where the action bar (see action bar below) menu items are defined.[3]

The bottom right hand corner of Figure 2 identifies the additional services declared in the manifest file that are needed by this application.

BlueTooth Link Connection Sequence and Program Flow

When the NimbleSig app is first started, the Bluetooth link to the NS3 module is not connected.

BLUETOOTH CONNECT REQUEST buttons are located on the tool bar, which is called the *Action Bar*, located along the top of the screen shot shown in Figure 3. When either of the CONNECT A DEVICE buttons, located on the right side of the *Action Bar*, are pressed the *select a device to connect* pop up window appears with a list of the previously paired Bluetooth devices. Note that the Bluetooth transceiver and the Bluetooth interface within the tablet must be paired prior to starting the NS3 controller app. This is done via the tablet *Settings* menu and will be described in detail in Part 3 of this article. The source code for this pop up window list is located in the *DeviceListActivity.java* file shown under services in Figure 2. In this case I had named

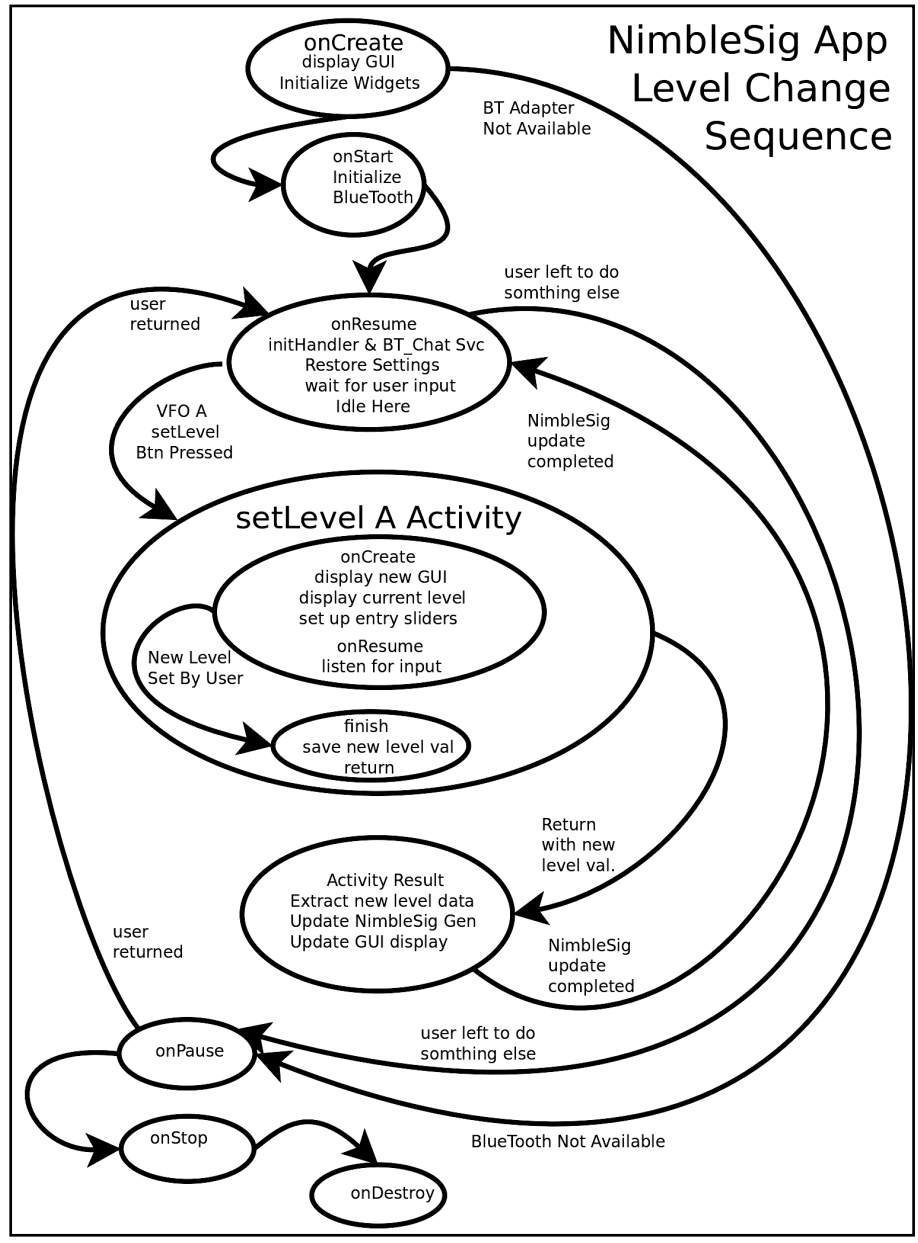


Figure 4 — State diagram — Set output level of VFO A function.

the NimbleSig module Bluetooth transceiver TABT01 and had previously paired it with my tablet. As shown, TABT01 appears as the bottom item in the list. When selected, the pop up list disappears and the wireless connection is typically established within a few seconds. Once connected, a connection status label will appear under the title in the *Action Bar* (see Figure 1) to confirm the link establishment. In addition, momentary pop up notification messages (accompanied with audible beeps if the tablet volume controls are turned up) appear, advising of the Bluetooth connection status changes. At this point the app is connected and ready to control the NimbleSig module.

Figure 4 is a simplified state diagram, which illustrates the program flow from initialization through to the execution of a basic user command. As shown, when the NimbleSig app is first started in response to the user's tapping of the *NimbleSig* icon on the tablet home screen, the *onCreate* method is called when the app is called by the Android operating system. The *onCreate* method first checks that the tablet is Bluetooth equipped, and if so it continues by building the GUI shown in Figure 1. Should the tablet not be equipped with the Bluetooth option or should the Bluetooth function be disabled, then the app is immediately aborted by branching down to the *onPause* method, followed by the *onStopped* and finally *onDestroy* lifecycle methods to close the app. It should be noted that all Android activities follow the *Created*, *Started*, *Resumed*, *Paused*, *Stopped* and *Destroyed* lifecycle steps, which are discussed in detail on the Android Developers website.[4]

Once the GUI is built and initialized by *onCreate*, control is passed to the *onStart* method, which initializes the Bluetooth link. Control is then passed to *onResume*, which initializes the Bluetooth service, restores settings from the previous session and makes the GUI graphics visible to the user.

The *onResume* method is in essence the idle loop for the app. The program sits here awaiting commands from the user or data from the Bluetooth link. The RF power meter is polled periodically to provide a continuous RF power detection update. Should the user decide to do something else with the tablet, such as browsing the Internet, the NimbleSig app will be placed in the background. As long as the tablet doesn't need any of the resources used by the NimbleSig app, which is typically the case, then the NimbleSig app will remain intact until the user re-selects this app, which brings it back to the foreground. When the user returns to this task the system typically re-enters where it left off via the currently focused *onResume* method.

If the user presses the VFO A LEVEL button, when back at the main GUI (Figure 1), the *setLevelA* activity is called via its *onCreate* method. The associated program flow for this action is shown in the center of Figure 4. When the *setLevelA* activity *onResume* lifecycle method is executed, a smaller GUI user data entry window pops up into the foreground as shown in Figure 5. Both course and fine volume control type sliders are provided for setting the level. In this case only two sliders are needed as the range of adjustment is just from –10.0 to –20.0 dBm. The fine adjustment easily accommodates setting the level in 0.1 dB increments.

Ratchet Slider Tuning

For other functions with a wider adjustment range requirement more sliders are provided. For example the modulation frequency range is 1 Hz to 20 kHz with a 1 Hz resolution. In this case, as shown in Figure 6, four sliders are provided. The top slider covers the whole 0 to 20 khz range in 1 khz steps while the finer sliders have resolutions of 100 Hz, 10 Hz, and 1 Hz, which have a range of ± 1 kHz, 100 Hz and 10 Hz respectively. Note that the finer sliders are elastic in feel and automatically hop back to center scale when the touch is lifted. The course slider is first used to set the approximate frequency. The finer sliders are then used in a progressive manner to set the exact frequency down to 1 Hz resolution. Since these self centering, finer resolution sliders pop back to center when released they can be used quite conveniently to "ratchet" the frequency up or down.

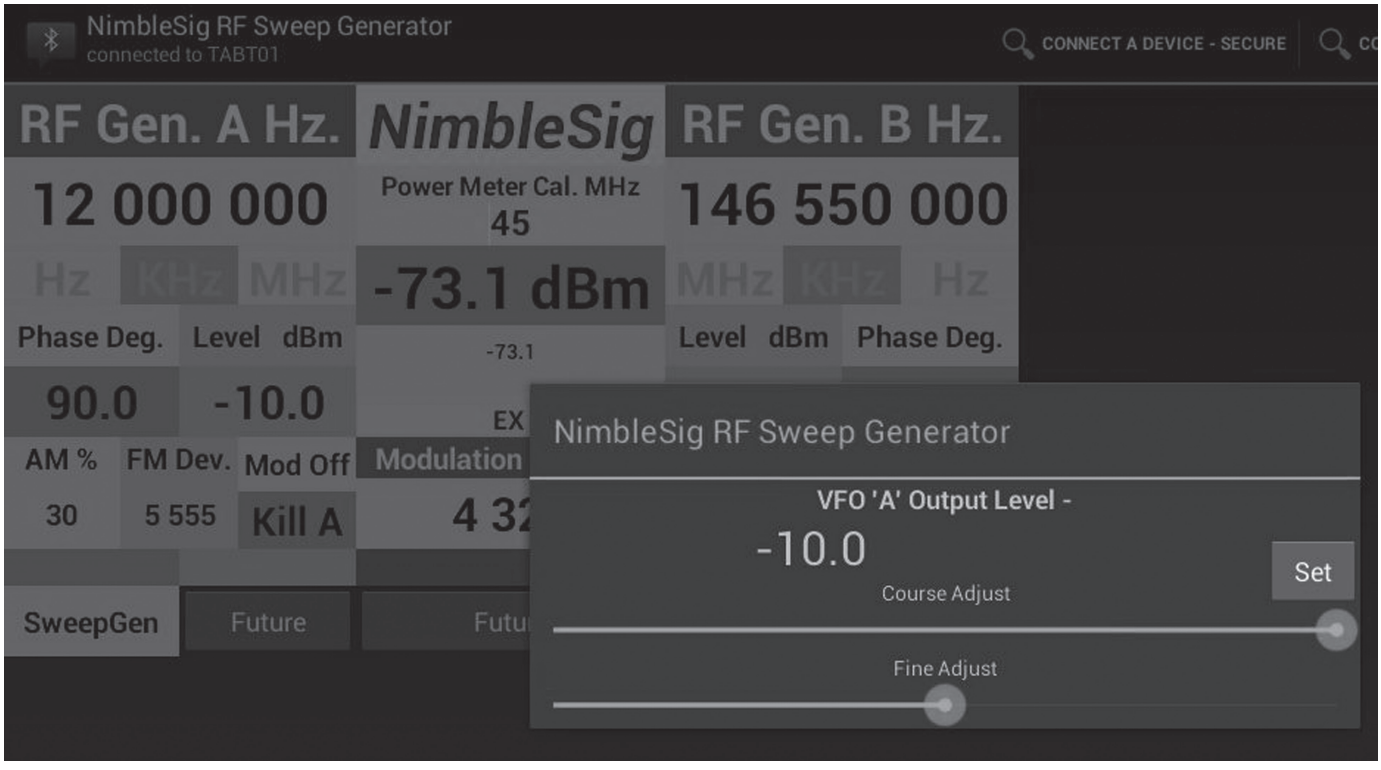


Figure 5 — VFO A output level adjustment sliders.

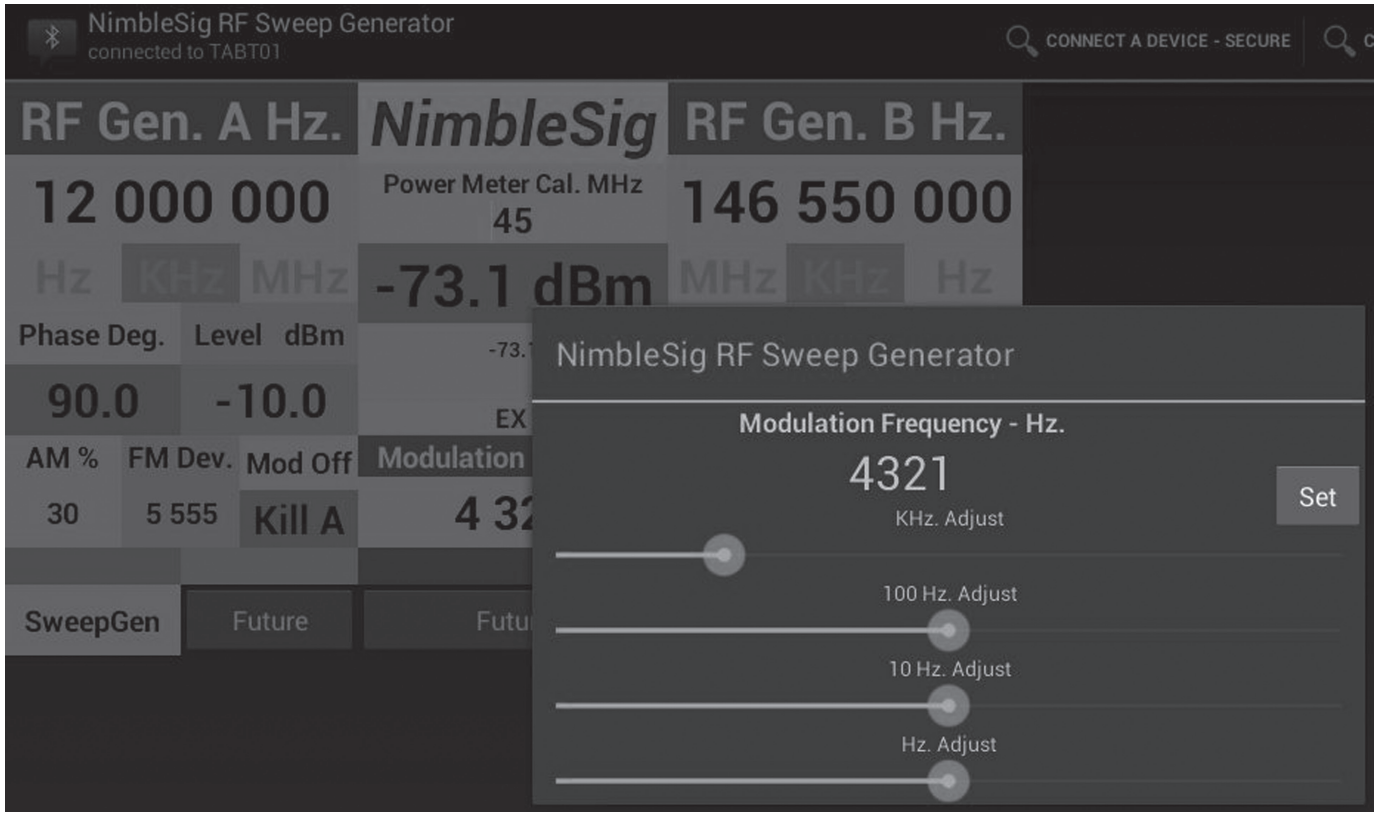


Figure 6 — Modulation frequency adjustment slider bank.

Once the frequency is set to the desired value, pressing the *Set* button exits the setting change activity, and the new value is returned back to the main activity by calling the *onActivityResult* method. The *onActivityResult* method extracts the requested new value, sends the new value to the signal generator module and updates the main GUI display. Once complete, the program flows back to the *onResume* method, where the main activity returns to the idle state. The program flow is similar for the majority of the buttons.

RF Carrier Frequency Entry

The method of entry for the RF carrier frequencies is an exception to the use of ratchet sliders. In these cases the built in capability of the Android *Edit* widget is used. For these entries, you double tap the existing frequency value and the keyboard pops up allowing direct digit entry. To switch to a specific frequency, enter the desired value in either MHz, kHz or Hz resolution, and then presses the corresponding MHz, kHz or Hz button. This saves key strokes. So to go to 10 MHz all you need to type is "10" and then press the MHz button. Similarly to switch to a frequency with kHz resolution, such as 14,140 kHz, just type 14140 and press the kHz button.

NimbleSig III Sweep Generator

Figure 7 is a screen shot of a 45 MHz IF filter frequency response. The Android tablet lends itself nicely to the provision of a self-documented grid display for swept frequency measurements. As shown there is room to label the screen with all the significant sweep generator parameters. Start, stop and center frequency as well as the full span width and the span per horizontal division parameters are all labelled. The additional *Step Frequency* and *Step Level* parameters change dynamically as the sweep progresses. The *Step Level* of –70.2 dBm shown in Figure 7 corresponds

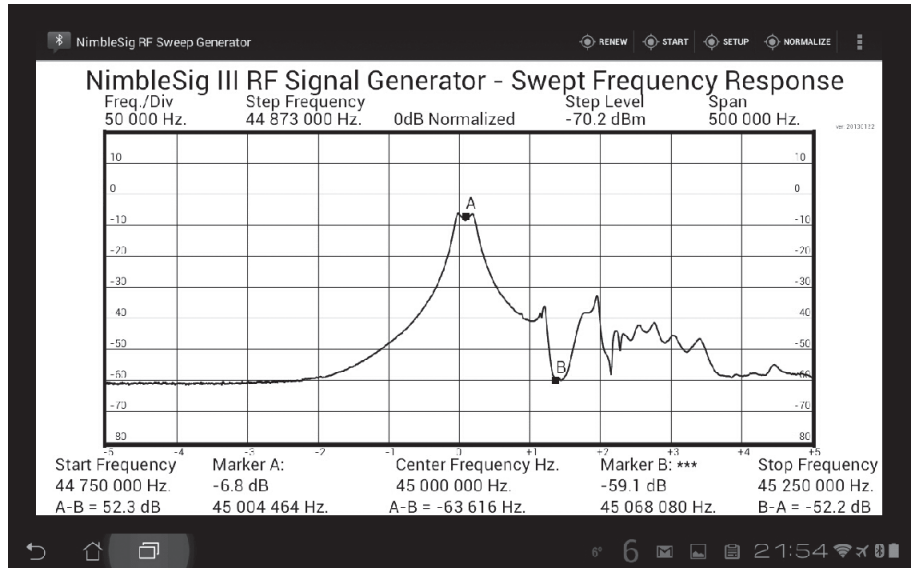


Figure 7 — Swept frequency response of a 45 MHz IF crystal filter intended for a NBFM application.

to the level that was measured when the generator Step Frequency was at 44,873 kHz during the frequency sweep. Note that the sweep response shown in Figure 7 has been normalized to directly show filter insertion loss in dB. In contrast, the *Step Level* value is the absolute power level that is presented in units of dBm. This explains why the 44.8 MHz insertion loss trace level shown in Figure 7 is about 8 dB higher than the Step Level value.

Marker A in Figure 7 reports the near passband center frequency of 45,004,464 Hz with an insertion loss of 6.8 dB. Marker B shows a passband null in the 45,068 kHz region that according to the B – A calculation is 52.2 dB down from marker A. The frequency difference between the center of the passband and the null is shown at the bottom center of the screen as 63,616 Hz. The position of the markers can be changed just by touching the screen at the desired point. Once the user is ready, the screen can be captured by a long touch on the stack history widget located towards the left side of the bottom tool bar. The screen shot can then be shared with others via available connectivity options, such as e-mail.

Prior to the connection of the 45 MHz IF crystal filter used for the Figure 7 example, the sweep display had been normalized by looping the generator output directly back to the detector via the test cables. This calibration compensates for any loss in the test leads and normalizes the sweep reference to 0 dB. Thus the –6.8 dB value shown for marker A corresponds to the actual insertion loss of the filter within the passband.

Sweep Generator Setup GUI

Figure 8 is a screen shot of the sweep generator setup GUI. As shown, there are eight slider bars that provide a wide selection of adjustment resolutions. The slider bar colors follow the standard resistors color code according to coarseness. When a frequency button in the upper section of the screen is touched it is highlighted in green indicating it is selected for control. For example if the user were to touch the *Center Frequency Hz* button it would turn green, at which time the sliders could be used to set the center frequency.

The data entry for the *Sweep Steps* and *2 dB/Div Ref dBm* values are done using the pop-up keyboard instead of the sliders. The keyboard appears when either one of these buttons is touched, and once the desired value is entered the operation must be completed by touching the *done* key in order for the new value to be registered.

The *Sweep Steps* provides control over the size of the frequency steps and the speed of the sweep. Fewer steps results in a faster sweep, but an increase in step size reduces the horizontal resolution. If the steps are too large it is possible to miss a sharp change of frequency response. For example if there happens to be a sharp null in the frequency response that lies between frequency steps the null might be stepped over and consequently not displayed. With my tablet, a 200 step sweep, which is usually fine enough, takes less than 10 seconds but it is noticeably course. A 1000 step sweep takes about 45 seconds but provides a very nice high resolution trace. Thus, if I wish to save a response for documentation I usually do a 1000 step sweep. A setting less than 200 steps might be desirable when I need a quick, repetitive trace refresh to see immediate results while tuning a circuit for a desired frequency response.

The sweep generator span is set up by specifying the sweep width and the center frequency. The start and stop frequencies are then automatically calculated.

The *Scale dB/Div* button is simply a toggle that switches between either 10 or 2 dB per division. Figure 9 illustrates the close-up view obtainable with the 2 dB/

Figure 8 — Screen capture of the NimbleSig RF sweep generator setup GUI.

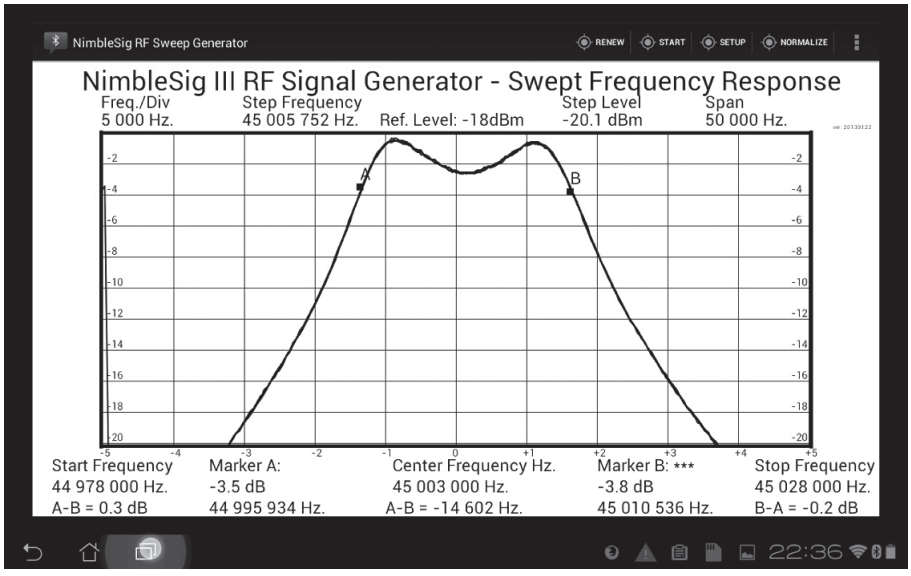


Figure 9 — Screen capture of a close-up swept frequency response of the 45 MHz IF crystal filter. Note that this sweep uses a 50 kHz span, compared to the 500 kHz span of Figure 7.

div setting. A downside of this close-up view is that the window dynamic range is reduced from 100 dB to 20 dB, so if the level reference is not set properly the trace may be off screen. You must set the 2 dB/div reference value to ensure the sweep appears were desired within the reduced dynamic range spectrum display grid. In Figure 9 the reference level is shown as –18 dBm. Note the normalization function is not provided for the 2 dB/div display, because the ability to set the reference level manually makes it unnecessary.

Source Code

The open source code files for this app have been posted on the ARRL *QEX* files website for freedom of use.[5] Note that although this application is fully functional, it is still under development and needs more work to smooth out the edges. There are no doubt bugs that will need to be ironed out once identified. If someone wishes to do something similar with an Android device, I hope that this code is of some value to use as a reference. It could be used in its present under-development state for controlling a NimbleSig III module if installed on a tablet similar to mine.

Part 3 to Follow

In Part 3 of this article, I will describe how an economical Bluetooth transceiver can be used to provide a wireless link for controlling a microcontroller based project.

Notes

[1]Thomas Alldread, VA7TA, "NimbleSig III — Parts 1, 2, and 3," *QEX*, Jan/Feb, Mar/Apr, May/Jun 2009. The articles and more details are available on the author's website at: **www3.telus.net/ta/NimbleSig%20III/**.

[2]Learn more about the manifest file at: **https://developer.android.com/guide/topics/manifest/manifest-intro.html**.

[3]For a tutorial on adding action buttons, go to: **https://developer.android.com/training/basics/actionbar/adding-buttons.html**.

[4]There is a tutorial about starting a lifecycle activity at: **https://developer.android.com/training/basics/activity-lifecycle/starting.html**.

[5]The program files for this project are all open source code files. They are available for download from the ARRL *QEX* files website. Go to **www.arrl.org/qexfiles** and look for the file **7x14_Alldread.zip**.

Don Dorward, VA3DDN

1363 Brands Ct, Pickering, ON L1V 2T2, Canada; **ddorward@sympatico.ca**

A Linear Scale Milliohm Meter; Another Look

With a few basic components you can build a dedicated milliohm meter.

In the July/August 2012 issue of *QEX*, Steve Whiteside, N2PON, described the design and construction of a stand-alone milliohm meter for general use.[1] I had been looking for something similar, to make it easier to check some ground connections and some coax connections on a group of older ARES 2 m antennas built for emergency deployment. I, too, had found that my trusty Fluke Digital Multimeter (DMM) was unreliable at the low end, for example when trying to measure 1 or 2 Ω or less.

Like Steve, N2PON, I wanted a no-nonsense, small, portable, milliohm meter that I could pick up anytime and put to use. Figure 1 shows my finished meter, ready to use.

My main design goals were:

1) Use standard components as much as possible.

2) Simplified design, with minimum component count.

3) Temperature/drift independence.

4) AA battery power supply, 3 V maximum.

5) Tolerant of up to 20% battery discharge.

6) Single linear scale, 0 – 2.0 Ω

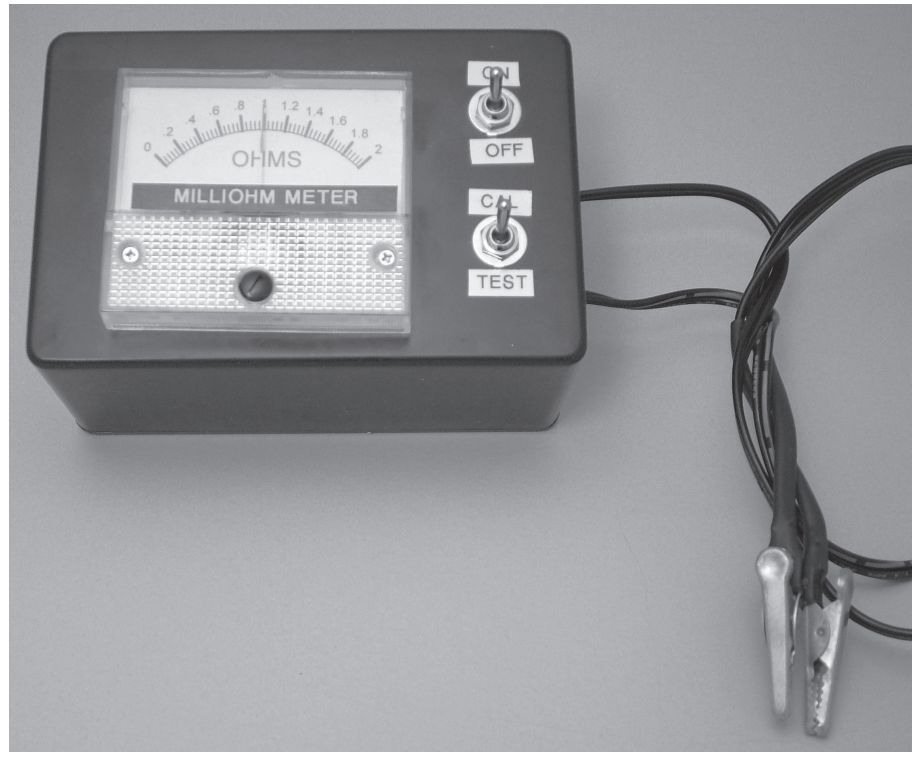


Figure 1 — Here is the completed milliohm meter, ready for use.

Design

At first thought, I planned to use the old reliable LM317T adjustable voltage regulator IC. One popular adaptation of this device is use it as a constant current source, and this is well described in various manufacturers' data sheets. For example, see Figure 2.

This simple circuit with just the two

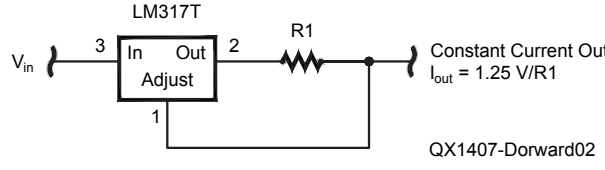


Figure 2 — This simple circuit uses only two components to produce a stable constant current source.

[1]Notes appear on page 26.

components shown, is capable of providing a surprisingly accurate and drift-free constant current source, despite varying input voltage or ambient temperature. The catch, however, is that the minimum input voltage, V_{in} for the LM317T is 3 V, most of which is lost in this configuration to the combination of drop-out voltage and the forced V_{ref} voltage of 1.25 V across R1.

I eventually chose to use the LM1117T, which is a low drop-out regulator, and has the same pin-out and TO-220 package as the LM317T.

The LM1117T has a specified maximum drop-out voltage of only 1.2 V. In this case $V_{drop\text{-}out} + V_{ref}$ is 1.2 V + 1.25 V = 2.45 V. Therefore, using a 3 V double AA cell power source, there is adequate margin for battery aging and the maximum 0.078 V full-scale meter voltage. I mounted the batteries on top of the case, as shown in Figure 3.

Figure 3 — Here is another view of the completed milliohm meter. The two AA batteries are mounted on the top of the project case.

The final circuit is shown in Figure 4. It consists of a simplified constant current source using the LM1117T. The constant current source drives the unknown resistance to be measured, and the resultant voltage drop is displayed on a small moving coil meter movement. Other than just the simplicity, an advantage of using this type of IC is the stability of the regulated output. Constant current variation due to temperature over the range of 0 to 50ºC is negligible. The circuit maintains regulation until the battery voltage drops to about 2.5 V, as tested on my bench.

The design actually starts with the meter movement that is chosen, and requires us to first determine the full scale voltage. This can be done in several ways, but I took the direct approach and used a digital voltmeter to measure the meter terminal voltage while I connected a 10 kΩ resistor in series with a 5 V adjustable power supply and the meter. I adjusted the power supply carefully so that the meter read its full scale value of 500 µA, and noted the DVM reading. With the meter movement I used, the full scale voltage was approximately 0.078 V.

Since I wanted my milliohm meter to have a 2.0 Ω full scale reading, the constant test current needed is calculated as:

I_{const} = 0.078 V / 2.0 Ω, or approximately 39 mA.

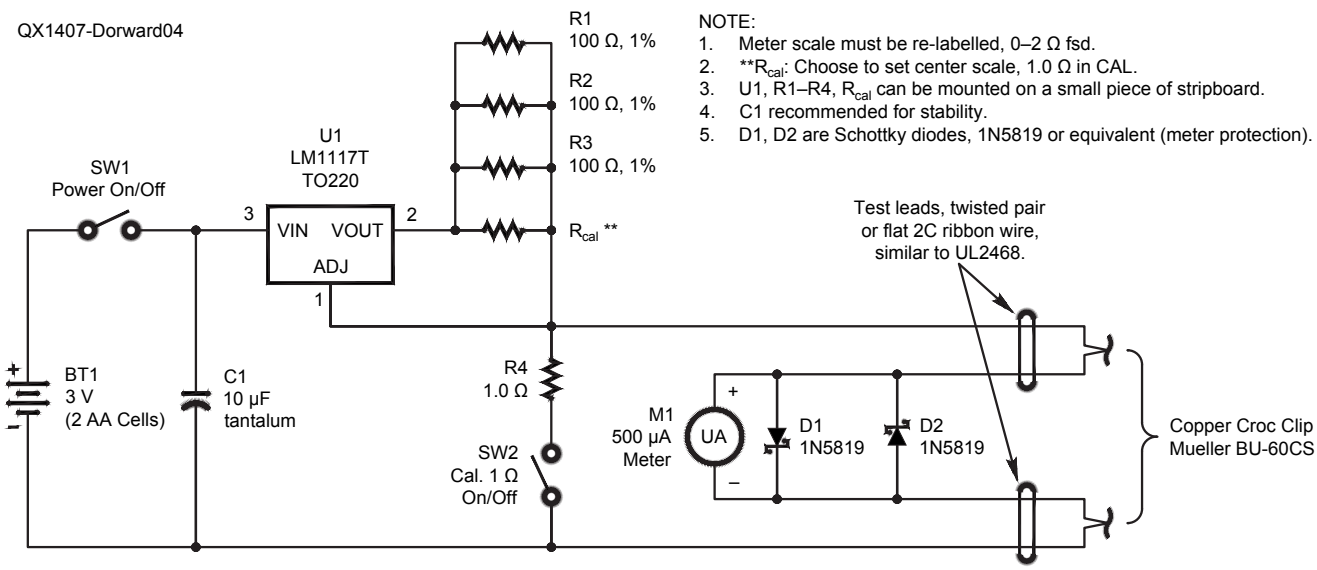


Figure 4 — This is the full schematic diagram of the milliohm meter circuit.

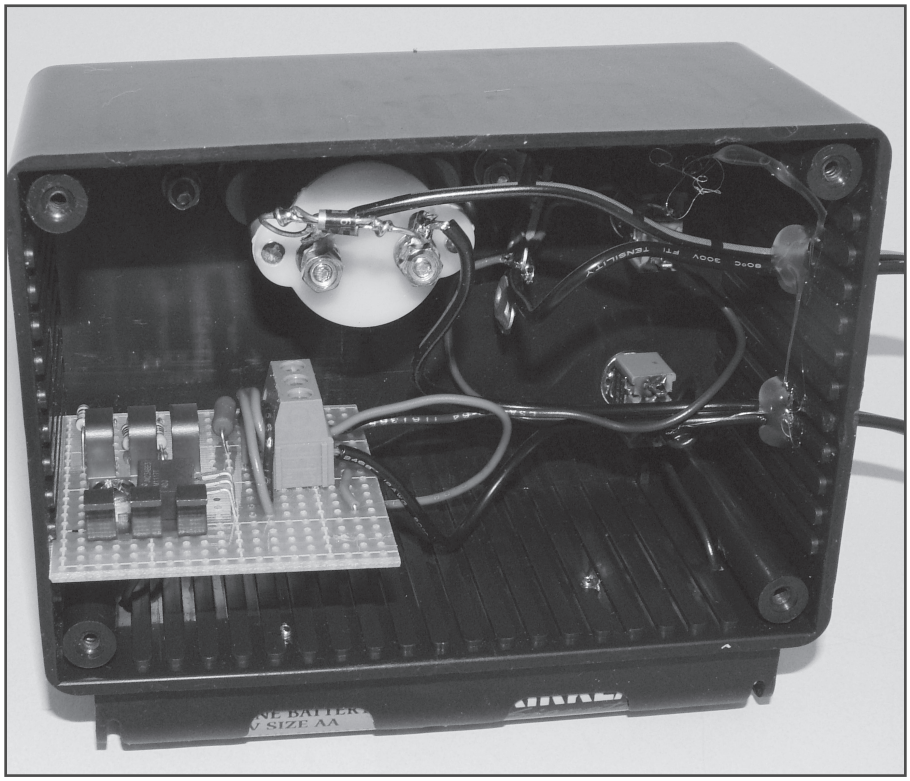


Figure 5 — This view into the back of the project case shows the circuit wired on the Veroboard. You can see the heat sink and LM1117T regulator on the left side of the board.

Construction Notes and Details

Many of the parts I found in my junk box, with the exception of the LM1117T. None of the other parts are especially critical. The regulator and the resistors are mounted on a small piece of Veroboard epoxy fiber copper strip board. Since the regulator power dissipation is very low, it doesn't really need a heat sink, but I thought it would be useful in assuring thermal stability. A square inch of aluminum would likely do just as well. Figure 5 shows a view of the construction inside the project box. You can see the Veroboard with the components, including the heat sink and LM1117T near the left side of the photo.

I included C1 on the schematic because the IC manufacturer recommends it, but I found the circuit operated fine without it. The photo also shows a screw terminal connector strip I mounted on the Veroboard, but direct wiring works too.

D1 and D2 are Schottky diodes with a voltage drop of about 0.30 V. They serve to protect the meter movement should you turn the power on with no test resistor or the Cal switch on.

The current setting resistors are made of three 100 Ω, 1%, ¼ watt metal film resistors connected in parallel, thereby acting as a single 33.3 Ω part. Without further adjustment, this would program a constant current of 1.25 V_{ref} / 33.3 Ω, or 37.5 mA.

Recall that we actually need about 39 mA, so it is necessary to add a fourth parallel resistor, which I called R_{cal}. The easiest way to find this value, is to connect a decade resistance box across the R1, R2, R3 combination, and switch the 1.0 Ω Cal test resistor into the circuit. Adjust the decade box for best center scale reading, and solder a resistor with that value across R1, R2, R3. In my unit, a value of 1000 Ω turned out to be perfect.

R4, the 1.0 Ω resistor used to check the 1.0 Ω center scale, is ideally a 1% part. I had difficulty finding such a part, however, and ended up using a 5% 1 W device I had on hand.

New Meter Scale

I laid out the custom scale with the easy to use program "Meter," from Tonne software.[2] Figure 6 shows the scale I created. Printed on good quality paper, it is glued to the back of the aluminum scale panel in the meter and then carefully re-installed.

Test clips

I first used a pair of ordinary plated-steel alligator clips from my junk box. They did work ok but I thought that for the long term, copper clips would be better. Figure 7 shows the clips I used. A ribbon wire pair was stripped, bare copper ends twisted, then inserted into the screw hole in the copper clip, and soldered.

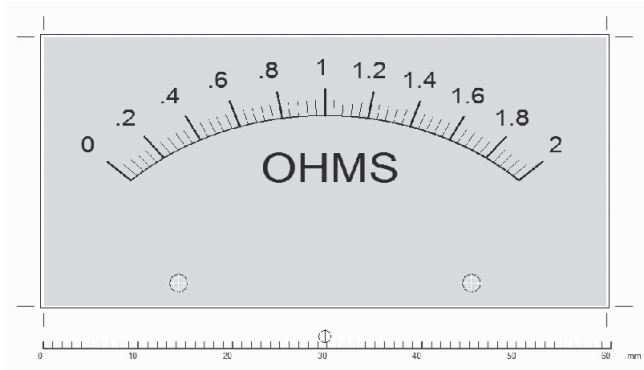


Figure 6 — This is the new meter scale printed using Jim Tonne's *Meter* software. See Note 2.

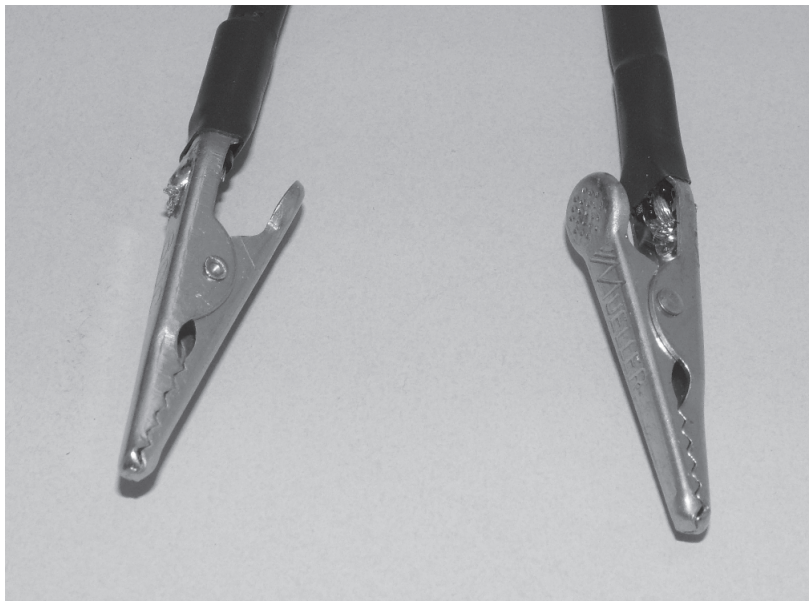


Figure 7 — The meter leads are soldered to the back ends of the alligator clips. You could also solder the wires closer to the mouth of the clips to reduce any resistance associated with the clips.

Table 1
Parts List

Parts used: Component	*Description*	*Source*
BT1 battery	1.5 V AA alkaline cells	Any good quality cell.
Battery holder	BC12AAL or equiv.	Digikey BC12AAL-ND
SW1, SW2	Mini toggle switch	3 A min., Silver contacts
Meter	0 – 500 μA, 2.5 inch wide	eBay
D1, D2	1N5819 Schottky diodes	Digikey, various mfrs.
C1 (optional)	10 μf, 10 V Tantalum cap.	Digikey 478-1838-ND
R1, R2, R3	100 Ω 1% Metal Film ¼ W	Digikey 100XBK-ND
R_{cal}	TBA (I used 1 kΩ, 5%, Carbon Film)	Junkbox
R4	1.0 Ω 1%, 1 W, Metal Film	VishayCPF11R0000FKEE6 Digikey CPF100CTR-ND
U1	LM1117T, Adjustable	Digikey LM1117T-ADJ/NOPB-ND
Heat Sink for U1	Avid, 20°C/W, 5070 type	Digikey HS112-ND
Alligator clips, 2	Mueller BU-60CS	314-1034-ND
Case	4.7 x 3.1 x 2.3 inches	Hammond 1591TSBK Digikey HM110-ND

This has been fine with the 2.0 Ω scale I used, which has a single "tic" resolution of 0.04 Ω. I suspect that if you choose to change or add a scale where the resolution is greater, say 0.01 Ω per "tic," you may want to consider separating the soldering point of the 2 leads, with the meter wires closer to the clip tips. Should you choose to use the steel clips instead, I would recommend this method of connection to minimize the small resistance of the clip in your measurements.

Last, a note on the wire used for the test leads. Wire gauge is not particularly important. I ended up using some two conductor zip wire cut-offs that came from a wall-wart power supply. They happen to be #18, and are fairly robust, but several wire gauge sizes smaller should work as well. Some might argue that a twisted pair might be better in avoiding unwanted noise pick up. I have not seen that as an issue.

Don Dorward, VA3DDN, is a 1963 Ryerson Electronics Technology graduate. Career positions have included: management of Research and Development, ISO9001 and ISO13485 Quality Systems, Regulatory Affairs, in the areas of electronic components and materials technical support, environmental testing and instrument calibration, automotive electronics product development, switch mode power supply development, medical electronics, UL/CSA and EU product safety testing and certification, including conducted and radiated EMI compliance.

Don developed programs for accelerated life testing methods such as HALT and HASS, in-house training for Quality Systems, ESD prevention, IPC Workmanship Standards for the Acceptability of electronic equipment. He shares 2 patents.

Don has been licenced as VA3DDN since 2002, with basic and advanced certification. He retired in 2006. He is a Life Member IEEE, a member of Radio Amateurs of Canada and an ARRL member.

Notes

[1]Steve Whiteside, N2PON, "A Linear Scale Milliohm Meter," *QEX*, Jul/Aug 2012, pp 33-38.

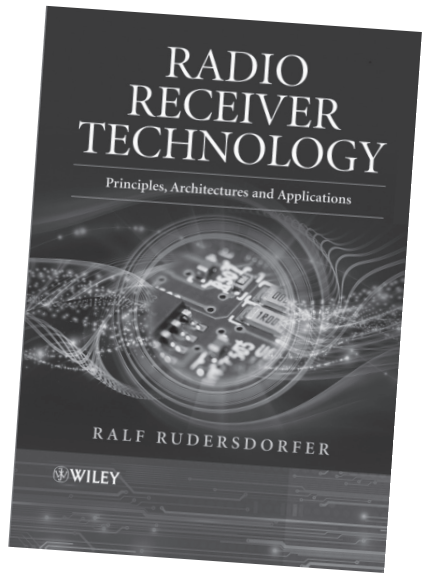
[2]James Tonne, W4ENE, *Meter* and *MeterBasic* are meter scale drawing programs available for download at **http://tonnesoftware.com/**. *Meter* requires the purchase of a text key to activate the program, while *MeterBasic* is free. The free version has some limitations, but is suitable for most simple meter scales. *MeterBasic* is included on the CD that comes with *The ARRL Handbook*.

New Book

Radio Receiver Technology: Principles, Architectures and Applications

Ralf Rudersdorfer, OE3RAA

In this book, the author introduces the reader to the basic principles and theories of present-day communications receiver technology. The first section of the book presents realization concepts at the system level, taking into consideration the various types of users. Details of the circuitry are described providing the reader with an understanding of fully digitized radio receivers, offering an insight into the state-of-the-art.



Key Features:

- ♦ Introduces the basic principles and theories of present-day technology
- ♦ Discusses concepts at system level (aligned to the various types of users)
- ♦ Addresses (fully) digitized radio receivers focusing on the state-of-the-art
- ♦ Close contacts to the industry were utilized to show background information
- ♦ Enables the reader to comprehend and evaluate the characteristic features and the performance of such systems
- ♦ Examines the entire range of parameters that are characteristic of the technology including the physical effect and measuring techniques
- ♦ Includes results of experiences gained in extended laboratory work and practical testing with examples
- ♦ Provides a uniform and systematic approach for ease of understanding e.g. many didactic figures for the visual illustration have been newly created as well as complete real-world examples

This book will be an excellent resource to understand the principles of work, for professionals developing and testing radio receivers, for receiver users (e.g. at regulatory agencies, surveillance centers, secret services, classical radio communications services), technicians, engineers and technicians who work with RF-measurement instruments, postgraduate students studying in the field and university lecturers. Chartered radio amateurs and handlers/operators will also find this book insightful. Due to high level of detail, it also serves as a reference.

Hardbackback
320 pages
November 2013
1SBN 978-1-118-50320-1
£81.951 €99.40 1 $130.00

Wiley-Blackwell E-Books and Online Books

Digital editions of the books featured are available for download to your computer or e-book reader.

Please visit **wiley.com** or your preferred e-book retailer for further details.

Scotty Cowling, WA2DFI

P O Box 26843, Tempe, AZ 85285; **scotty@tonks.com**

Hardware Building Blocks for High Performance Software Defined Radios

The author explores some alternative hardware for software defined radio use.

Low-cost, highly capable digital hardware is proliferating everywhere. Do names like Arduino, Beagle Board, Raspberry Pi, BeMicro or SoCkit mean anything to you? Is this some kind of secret code that has to do with very small Italian dogs that eat spherical fruit for breakfast? Not hardly! While the first three are names of low-cost embedded microcontroller (MCU) boards, the last two represent their counterparts in the world of programmable hardware. Wait, you say; microcontrollers *are* programmable hardware. This is true, but what makes BeMicro and SoCkit boards different is that they each contain a Field Programmable Gate Array (FPGA). See the "MCU Versus FPGA" sidebar for a look at the differences between an MCU and an FPGA.

An MCU executes instructions from a pre-defined instruction set in sequential order; the hardware is fixed, but the sequence of instructions is programmable. An FPGA, on the other hand, has no fixed instruction set or sequence of instructions, operates on data in parallel and has programmable logic and interconnections. Software defined radio (SDR) implementations can benefit greatly from the FPGA parallel hardware architecture.

And Now for Something Completely Different — A Software Defined Radio!

A few RF boards can be added to standard, off-the-shelf digital development kits to build a high-performance software defined radio. What these SDRs lack in polish, they make up for in performance. When you assemble a collection of boards, some of which were designed for a completely different purpose than building an SDR, what you end up with may not look pretty. If performance is your goal, however, you will not be disappointed.

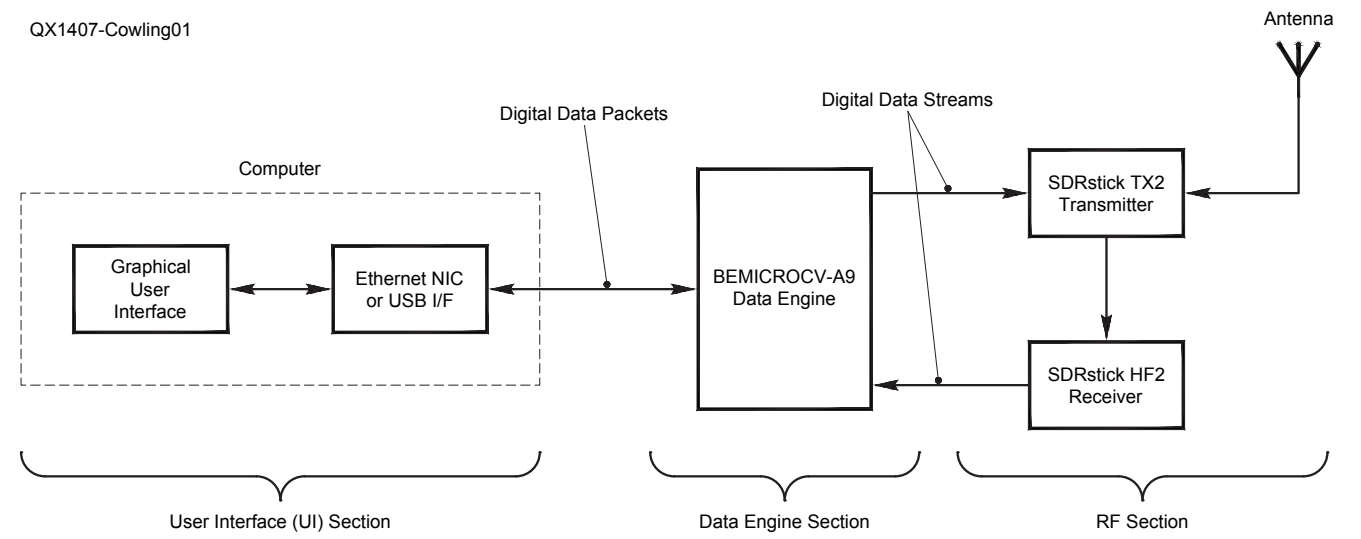


Figure 1 — An example of functional SDR sections.

I will present a hardware overview of the BeMicro series (BeMicro, BeMicroSDK, BeMicroCV and BeMicroCV-A9) FPGA development boards and the SoCkit System-on-a-Chip (SoC) FPGA development board, along with the SDRstick™ RF front-end boards necessary to transform them into a full-fledged digital down conversion receiver or a digital up conversion transmitter, or both at the same time!

Together, these boards make several different configurations of very high-performance digital down conversion/digital up conversion radios possible for only moderate cost.

System Functional Categories

We can break an SDR system down into three main functional categories (see Figure 1 for an example), starting from the operator and working our way toward the antenna:

1) User interface (UI)

2) Data engine

3) RF section (receiver front-end, transmitter strip)

This is definitely a simplified view, but still useful. Common SDR processes such as decimation, modulation and demodulation, digital filtering, data formatting and so on, are not necessarily confined to any one of the above broad functional categories. We will touch on these processes later on, but we will not dwell on the technical details. We will piece together our SDR and leverage the work others have already done to make it work.

The User Interface — 31 Flavors?

The user interface comes in many varieties: for different radios (such as openHPSDR, RF Space SDR-IQ™, FlexRadio Systems Flex-6000™); for different operating systems (*Windows*, *Linux*, *MAC OS*, *Android*); and for different specialized uses (such as GNU Radio, CW Skimmer, QtRadio). I have lumped these all into one category because they have one thing in common: they are the interface between the human and the radio. In reality, these software programs and the computers that run them perform many more tasks than just their one self-described function. In general, the user interface also performs modulation and demodulation, digital filtering, time-domain to frequency-domain conversion (for waterfall and panadapter displays), and control of all radio hardware and hardware DSP functions. As we shall see, some of these functions may be performed in the data engine, or split between the user interface and data engine. In either case, the functions are already coded and working. All we have to do is connect them properly.

The Data Engine — Heavy Lifting

The data engine is the bridge between two data domains. On one side is the freshly digitized data from the receive portion of the RF section or the digital data stream destined to be converted to analog RF in the transmit portion of the RF section. On the other side is the data to and from the user interface, which is typically some type of computer. Notice that there are no analog components to the data; the conversion between analog signals and digital data takes place in the RF section (and sometimes in the user interface section in the case of microphone or receiver audio). The data engine deals strictly with digital data.

Since the data coming from the RF section (samples from the receiver's ADC, for example) is in a different format than the data that is destined for the user interface section, the data engine must perform this conversion. The same is true for the outgoing data stream coming from the user interface section and ultimately destined for conversion to an analog transmit signal by the transmit DAC. The communications interface on the user interface side of the data engine in common SDRs today is either USB or Ethernet. Both USB and Ethernet use packets to transfer data, and since the raw data from ADCs or to DACs are in streams, packetization and de-packetization of the data must also be performed by the data engine.

The data engine may simply re-format raw data into packets (or packets into raw data), but often times the available data bandwidth on the user interface side does not match the required bandwidth on the RF section side. In the case of a low rate receiver ADC sending data over a high speed interface to the user interface, or a high speed interface from the user interface sending data to a low rate transmit DAC, the solution is easy since the packetized nature of USB and Ethernet allow idle periods where no data is sent. In the opposite case (high rate ADC to low speed interface or low speed interface to high rate DAC), receive data will be lost (user interface over-run), or the transmitter DAC will be starved for data (RF interface under-run). For example, in the HF2 digital down conversion receiver described later, the 16-bit ADC clocked at 122.88 MHz produces almost 2 Gbit/s of raw data. This data rate far exceeds the capacity of both high speed USB 2.0 at 480 Mbit/s and Gigabit Ethernet at 1 Gbit/s. The fact that both USB and Ethernet have packet overhead only makes the problem worse. The solution is implemented in the data engine hardware.

In the receive path, the data engine simply throws some of the data away. This is done in the digital domain by a process known as decimation. The key to building a useful receiver is to know which data to keep. That is determined by the center frequency of the receiver display bandwidth. This is the digital equivalent of the local oscillator in a super heterodyne receiver. An explanation of the mathematics behind decimation is beyond the scope of this article, but there are excellent references.[1, 2]

In the transmit path, the raw data from the user interface is digitally up-converted to a full-bandwidth data stream and sent to the DAC in the RF section. This is accomplished using digital mixing, again determined by the center frequency of the transmit bandwidth. This center frequency is analogous to the local oscillator in a mixer type transmitter.

The RF Section — Still Analog After All These Years

The RF section is the interface between the outside world of RF and the digital world of, well, the rest of the radio! Any receiver front-end protection, filters or attenuators are included in this section, along with transmitter filters, power amplification and antenna switching. Most of these functions may also be performed in the digital portion of the SDR, at least to a limited extent. There are some things, however, that must be done in the analog hardware. An SDR is supposed to be a *digital* radio, so why can't we do everything *digitally*?

One reason is simple, and it also applies to conventional all-analog radios. *The maximum ratings of the devices connected to the antenna port of the radio must not be exceeded.* The other reason applies to SDR receivers, but generally not to all-analog receivers. *Any signals above the Nyquist frequency must be prevented from reaching the input to the receiver ADC.* (The Nyquist frequency in a digital down conversion receiver is half the ADC sample clock frequency.) What happens when either of these rules is violated depends on the particular SDR design, but performance is invariably compromised in some way.

Electrostatic discharge protection devices should be used at the antenna input to protect front-end components, and internal analog attenuators can prevent large input signals from overloading analog low-noise amplifiers (LNAs) or ADC inputs. Of course, attenuators reduce the amplitude of all signals, making them less useful than filters in some situations.

There are two reasons to use analog filters in an SDR. The first and foremost is to prevent any components in the front end from saturating, including active attenuators, low-noise amplifiers or ADC inputs. Once saturation occurs, devices behave in a non-linear fashion; no amount of digital

[1]Notes appear on page 40.

processing is likely to correct this. Filters used for this purpose can be either internal or external, or both. SDRs operated in the presence of strong in-band signals (such as near a powerful broadcast station) or connected to very large antennas may need external filters to prevent front-end overload, but many SDRs need no front-end overload filtering at all.

The second reason to use analog filtering is to prevent any signals above the Nyquist frequency from reaching the input to the ADC. Image signals above the Nyquist frequency will fold back, or alias, onto signals below the Nyquist frequency. Once this happens, the two signals (the desired signal and the image signal) will become indistinguishable from each other. This type of filter is called an anti-aliasing filter, and is most often internal to the SDR.

Data Engines

The SDRstick™ RF boards currently support three different data engines, and more if you include other FPGA development kits with HSMC expansion connectors. The data engines that I will discuss are all manufactured by Arrow Electronics.

BeMicro — Not Quite Enough

The introduction of the first Field Programmable Gate Array (FPGA) System Development Kit created the potential for inexpensive, easy to build (read: plug together) Software Defined Radios. While I can only guess at which one was *the* first one, the BeMicro FPGA development kit (Figure 2) was one of the first.

The original BeMicro FPGA development kit used an Altera EP3C16 Cyclone III FPGA containing about 16 K logic elements (logic elements). The small BeMicro PCB also sported a 16 MHz clock oscillator, 256 K × 16 bit static memory (SRAM), 8 user-programmable LEDs, a USB programming/power port and an 80 pin micro-edge connector (MEC) for I/O. Arrow designed the BeMicro as a showcase for the Altera NIOS II embedded soft-core CPU (see the side bar), and even provided lab materials with design examples to help beginning FGPA users get started quickly. The design tools are all free and downloadable from the Altera website: **www.altera.com/products/software/sfw-index.isp**. The major drawback of the original BeMicro was the lack of any kind of high-speed communications interface. The only two external interfaces on the BeMicro are the USB port and the 80 pin micro-edge connector.

The BeMicro USB interface uses an FTDI interface chip. The FTDI interface chip can only operate at USB low or full-speed bit rates (1.5 or 12 Mbps, respectively). It connects directly to the dedicated FPGA serial programming (JTAG) port rather than general purpose FPGA pins. This makes it difficult for the FPGA programmer to use the USB port for a communications interface.

The BeMicro general-purpose I/O interface on the 80-pin micro-edge connector could be used for a high-speed communications interface. As you will see later, however, we need all 80 of these pins for the interface to the RF portion of our SDR. All in all, the BeMicro qualifies as a resounding "almost enough" hardware platform.

Along comes BeMicroSDK

The successor to the BeMicro, called the BeMicroSDK (the suffix stands for System Development Kit), added enough functionality to the somewhat limited original BeMicro to advance it from the "almost enough" to "now it is possible" SDR data engine category.[3] Let's see what the designers added to make BeMicroSDK (Figure 3) a viable SDR data engine.

For starters, BeMicroSDK sports a 10M/100M Ethernet port. We now have a relatively high-speed communications interface, at least compared to full-speed USB. Ethernet supports routable data packets and is as ubiquitous as USB on modern desktop and laptop computers. The memory size is increased from 512 Kbytes to 64 Mbytes, and changed to DDR SDRAM. A micro-SD memory card socket and a temperature sensor have been added, as well as three push-button and two slide switches. These last few items are not of much use to us in building an SDR data engine, but the Ethernet and DDR memory sure are! In the excitement, I almost forgot to mention the FPGA upgrade: the 16 K logic elements Cyclone III has been replaced by the newer and larger 22 K logic elements Cyclone IV, an EP4CE22 device. The BeMicroSDK became the data engine for the first SDRstick™ SDR.

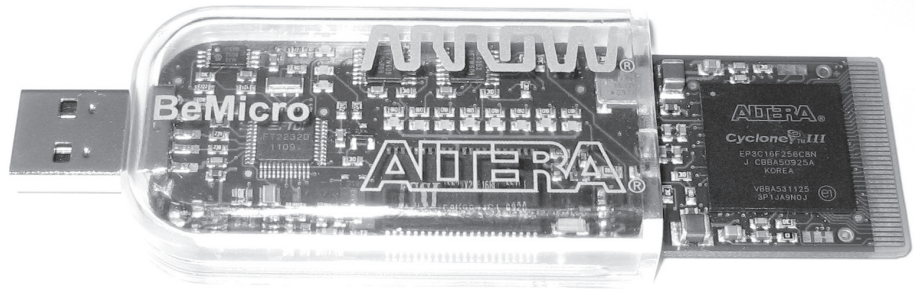


Figure 2 — The Original BeMicro board — not quite SDR material.

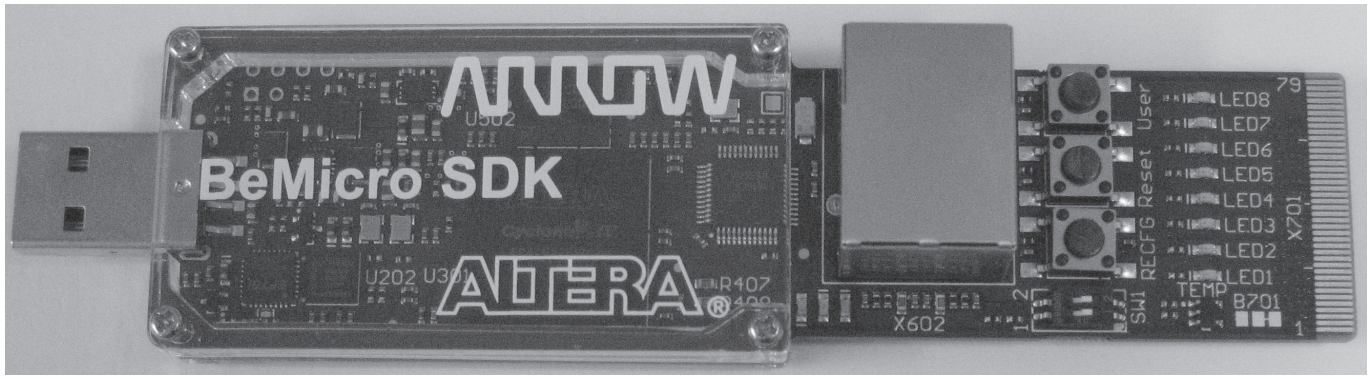


Figure 3 — The BeMicroSDK, with an Ethernet port, makes a good SDR data engine.

BeMicroCV-A9 - More Logic Elements, Anyone?

The evolution of the BeMicro line continued with the introduction of the BeMicroCV (where the suffix stands for Cyclone V).[4] While this is an interesting board (see Figure 4), in many ways it is a step backwards from the BeMicroSDK. It has a newer, but barely larger 25 K logic elements Cyclone V 5CEA2 device in place of the older Cyclone IV. The biggest problem for SDR use is the substitution of 64 GPIO pins on two 40 pin headers for the Ethernet port. While lots of I/O pins are useful for many things, even 64 of them cannot make up for the loss of the Ethernet port. If the BeMicroCV is not suitable for use as an SDR data engine, why do I even mention it? The answer lies in the successor to the BeMicroCV, called the BeMicroCV-A9.

The BeMicroCV-A9 (Figure 5) is different from the BeMicroCV in only two ways, but these two differences make the A9 (as it is called for short) an ideal SDR data engine. The first change is the re-purposing of 19 of the GPIO pins to add a Gigabit Ethernet port. This is a blazing fast channel for communicating with the user interface. The other change is to the FPGA. The A9 board uses (not surprisingly) a Cyclone V 5CEA9, with *301 K logic elements!* It is the largest member of the Cyclone V CE series, and perfectly suited to the job of SDR data engine. Now that we have two candidates for the job of data engine, you might think that we can move on to the RF section. But wait, not so fast; we have one more board that has something to offer the SDR builder.

CPU+FPGA - SoCkit to me!

There is more to life than bread alone, and more to an SDR data engine than just the raw number of logic elements in its FPGA. There is a new kid on the block that offers the best of both the FPGA world and the MCU world. It is called the system on a chip field programmable gate array, mercifully abbreviated SoC FPGA. This relatively new class of programmable device integrates both an FPGA and an ARM processor on the same silicon chip. The processor portion of Altera's SoC FPGA is called the hard processor system, or HPS for short. The term *hard* refers to the fact that the processor is hard-coded into the silicon, and not built by interconnecting programmable logic elements the way a soft-core processor is made. The hard processor system runs much faster than a soft-core processor and requires a smaller area on the silicon die. The hard processor system of the Cyclone V SoC FPGA in our next data engine runs at 800 MHz, whereas most soft-core processors are limited to a clock rate of around 300 MHz.

In a clever combination of the SoC abbreviation and the last word in the term system development *kit*, Arrow Electronics' SoC FPGA board is called SoCkit.[5] This somewhat obscure reference to a 1960s TV comedy show sketch does not escape older readers, I am sure. Young squirts can check Note 6.

The SoCkit board is built on a much larger form factor than the BeMicro series of data engines (see Figure 6). SoCkit's larger physical size allows room for far more features than we have seen so far on our candidates for SDR data engine duty. Here is a partial feature list:

- Altera Cyclone V SoC FPGA with 110 K logic elements and dual-core ARM Cortex-A9 CPU
- Two banks of DDR3 SDRAM (2 GByte total)
- Gigabit Ethernet port
- High Speed Mezzanine Connector (HSMC) expansion connector
- MicroSD Card connector
- USB serial port
- USB 2.0 OTG port
- USB Blaster programmer
- 128 × 64 pixel LCD display
- Video DAC with VGA connector
- Audio CODEC with line out/line in/mic in connectors
- 8 each: LEDs, pushbutton switches, slide switches

This is an impressive list of features for a board that costs under $300![7] Although Gigabit Ethernet is included, as well as a microSD card socket, there is no micro-edge connector like there is on every BeMicro series board. We will have to connect our RF boards to the high speed

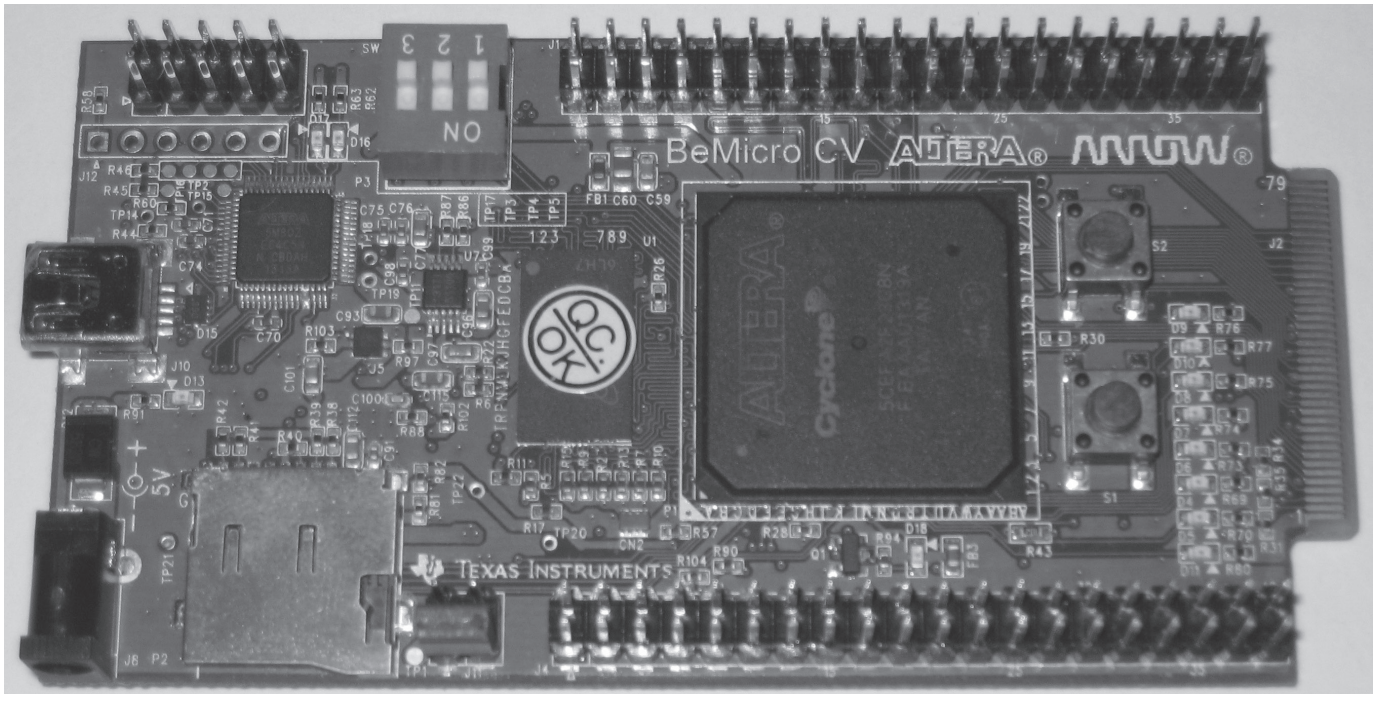


Figure 4 — BeMicroCV — No Ethernet port makes this one a non-starter for SDR.

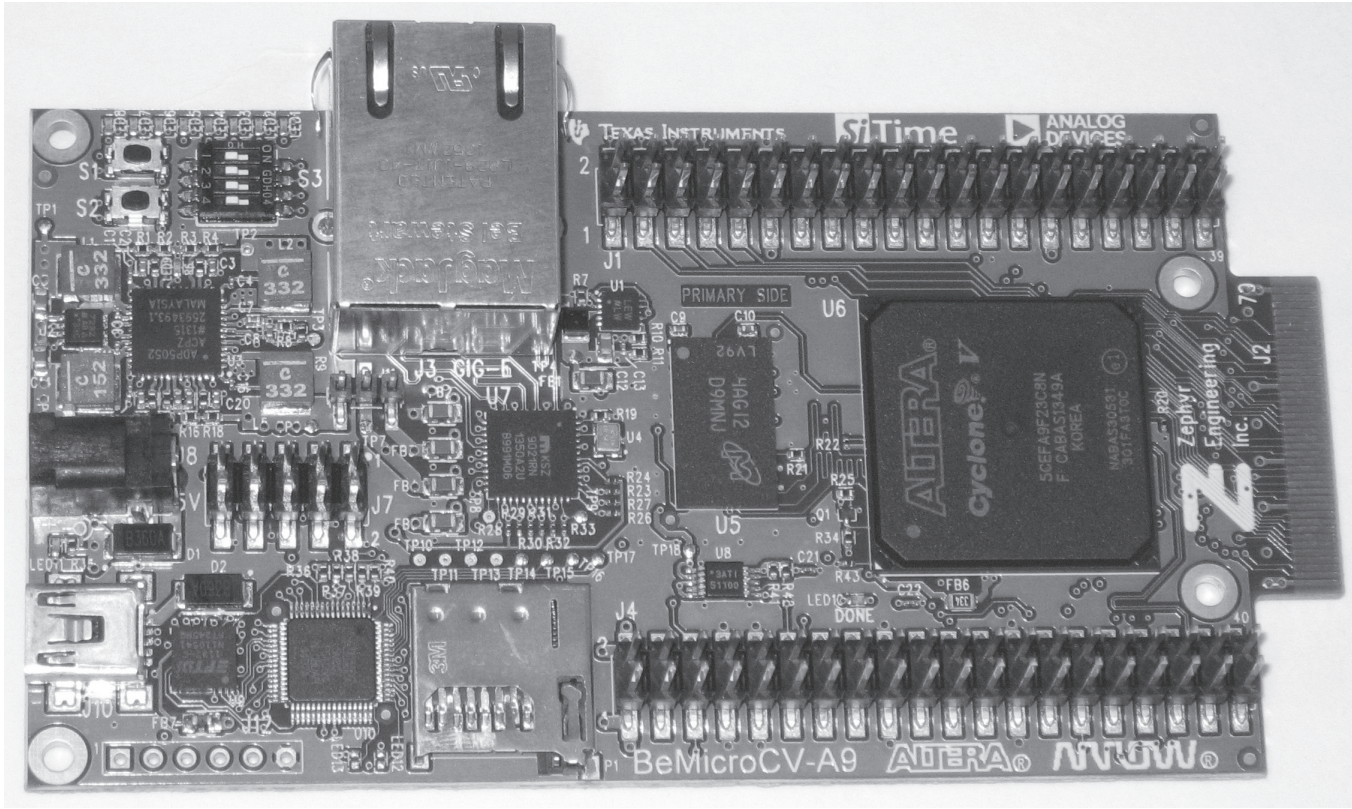


Figure 5 — BeMicroCV-A9, The Ultimate SDR data engine?

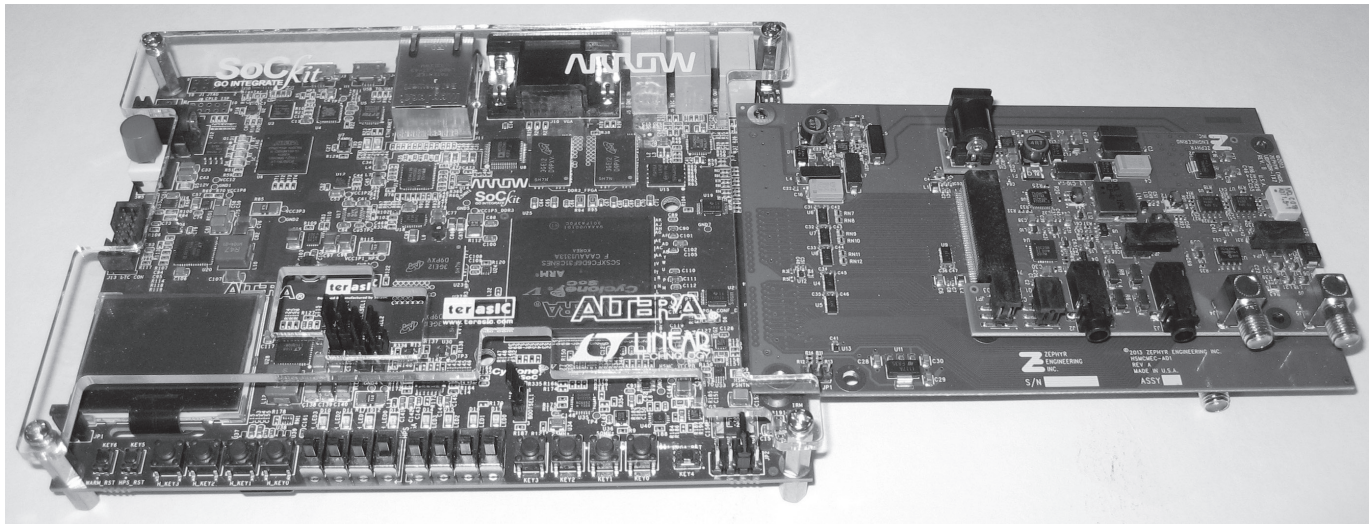


Figure 6 — SoCkit board with AD1 adapter, HF2 receiver and TX2 transmitter.

mezzanine connector (HSMC) instead. The HSMCMEC-AD1 adapter card (AD1 for short) was designed for this specific purpose (see Figure 7). The AD1 adapter plugs onto the high speed mezzanine connector port of virtually any development board (many off-the-shelf FPGA development boards have them) and provides two micro-edge connectors, one male and one female. The male, or board-edge connector accepts HF1 or HF2 receiver boards, while the female socket accepts the TX2 transmitter board. (More on the RF boards in the next section.) The AD1 performs level translation, since high speed mezzanine connector voltage levels can be 1.8 V, 2.5 V or 3.3 V, while the micro-edge connector RF boards all use 3.3 V signaling. The AD1 also provides separate receive and transmit connectors, allowing the TX2 transmitter to be used with the HF1 or HF2 receiver boards. Speaking of receivers, let's move on now to discuss the RF section of our SDR.

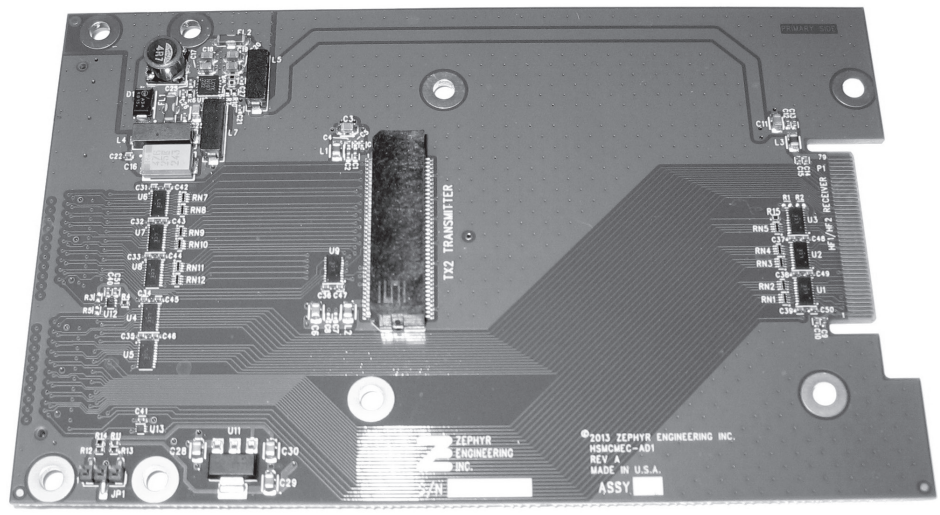


Figure 7 — The AD1 high speed mezzanine connector to micro-edge connector adapter board.

Receivers

There are two models of SDRstick™ micro-edge connector-compatible receiver boards: UDPSDR-HF1 and UDPSDR-HF2. The less expensive HF1 front-end (Figure 8 and Figure 9) performs decently, and employs a 14 bit ADC sampling at 80 mega samples per second. Paired with a BeMicroSDK or BeMicroCV-A9, the HF1 makes a fully functional 100 kHz to 30 MHz receiver. The TX2 transmitter cannot be used to make the system into a transceiver, since there is no micro-edge connector expansion connector on the HF1 receiver.

The high-performance HF2 receiver (Figure 10 and Figure 11) incorporates the same components and a similar architecture to the receive section as the openHPSDR Hermes board.[8, 9] A Crystek CVHD-950 extremely low phase noise oscillator clocks the 16-bit LTC2208 ADC at 122.88 MHz. A programmable 0 to 31 dB step attenuator (Mini-Circuits DAT-31-SP+) ahead of the LTC6400-20 20 dB gain differential ADC driver helps prevent overload. The HF2 receiver board has a transmit expansion micro-edge connector for the TX2 transmitter board described in the next section. The HF2/TX2 combination (along with a suitable data engine) makes a complete transceiver. Table 1 shows a comparison of the features of the two receiver boards.

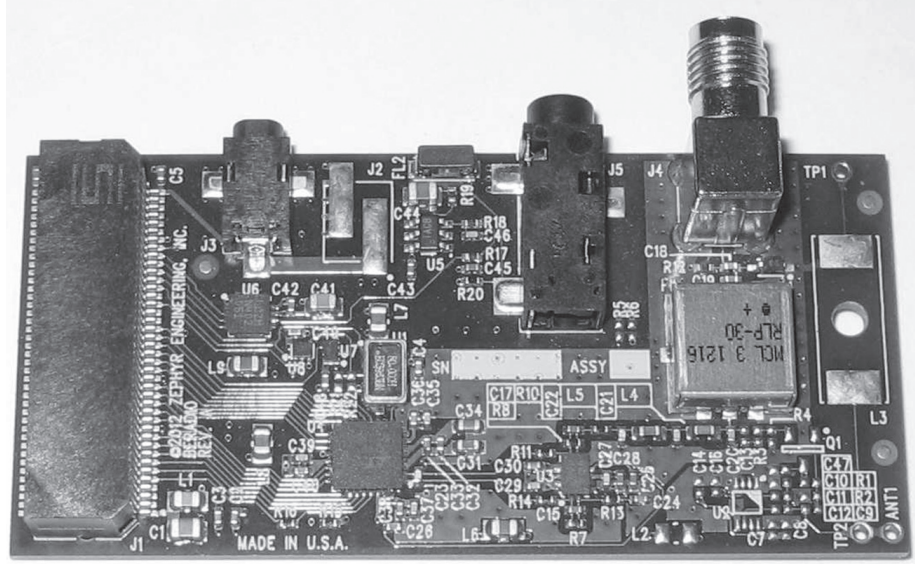


Figure 8 — The HF1 100 kHz to 30 MHz receiver.

Transmitter

The TX2 transmitter (Figure 12 and Figure 13) uses the same components and a similar architecture to the transmit section of the openHPSDR Hermes board (see Note 9). The TX2 uses an Analog Devices

What is a Virtual Receiver?

SDR receivers are built from parallel FPGA logic and software programs that perform mathematical operations (digital signal processing, or DSP) on a raw stream of data from an ADC. The processed digital data stream can be routed to client software for further processing or converted back to the analog domain to drive a speaker. The same raw data stream may feed multiple instances of DSP hardware and software, and each DSP instance can be independently configured (for example, center frequency, modulation type, filtering, and so on). Each one of these instances of DSP hardware and software is called a virtual receiver (FlexRadio Systems calls it a "Slice Receiver.") It is *virtual* because the receiver exists as a series of programmed processes rather than analog hardware, such as mixers and oscillators. Even though it is *virtual*, a virtual receiver is still implemented in hardware; it uses digital logic in place of analog components.

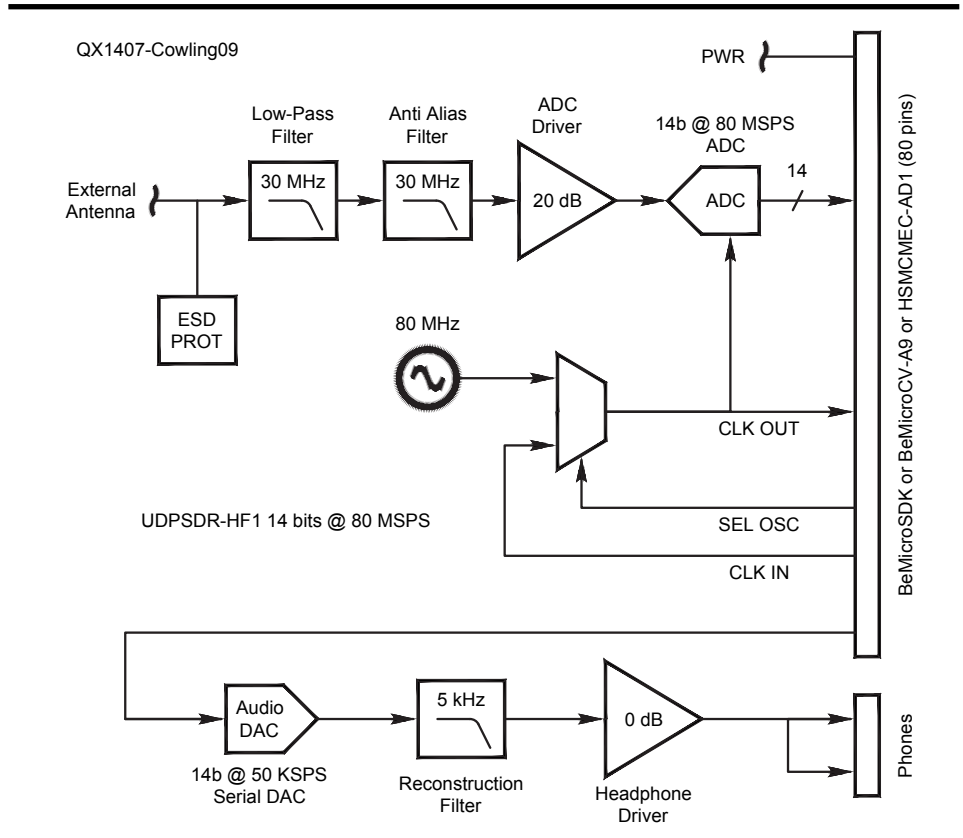


Figure 9 — HF1 receiver block diagram.

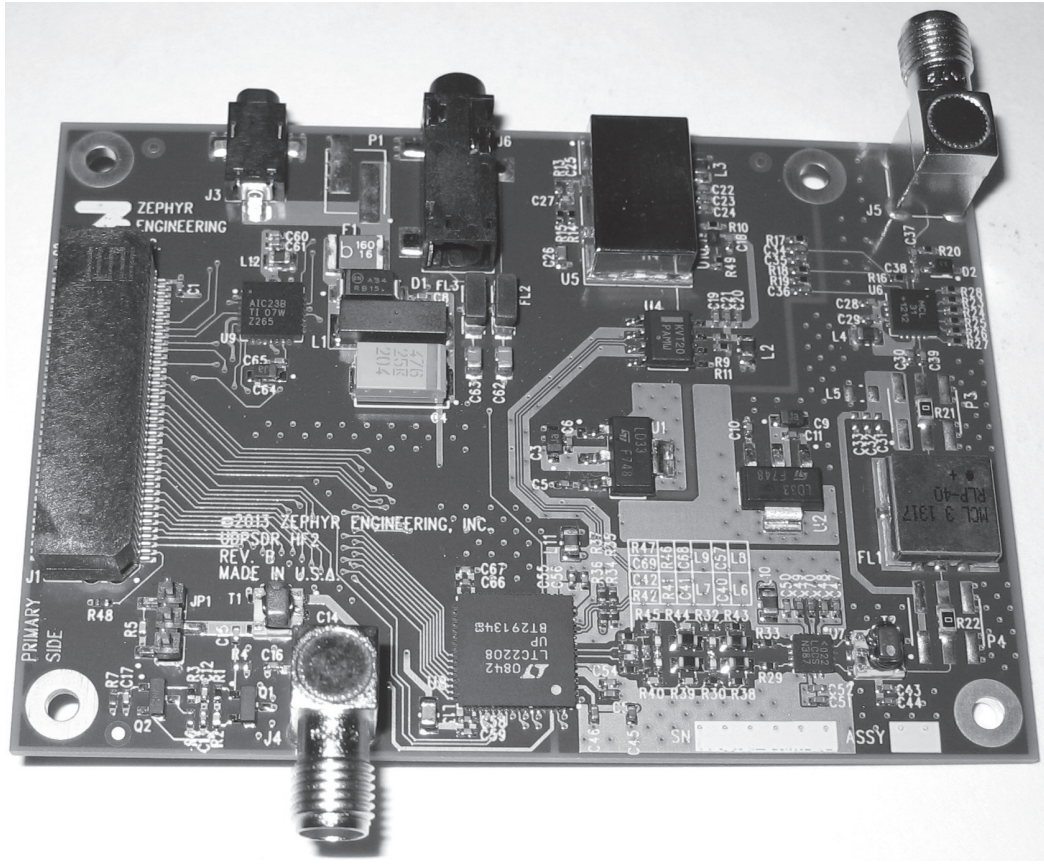


Figure 10 — The high-performance HF2 100 kHz to 55 MHz receiver board.

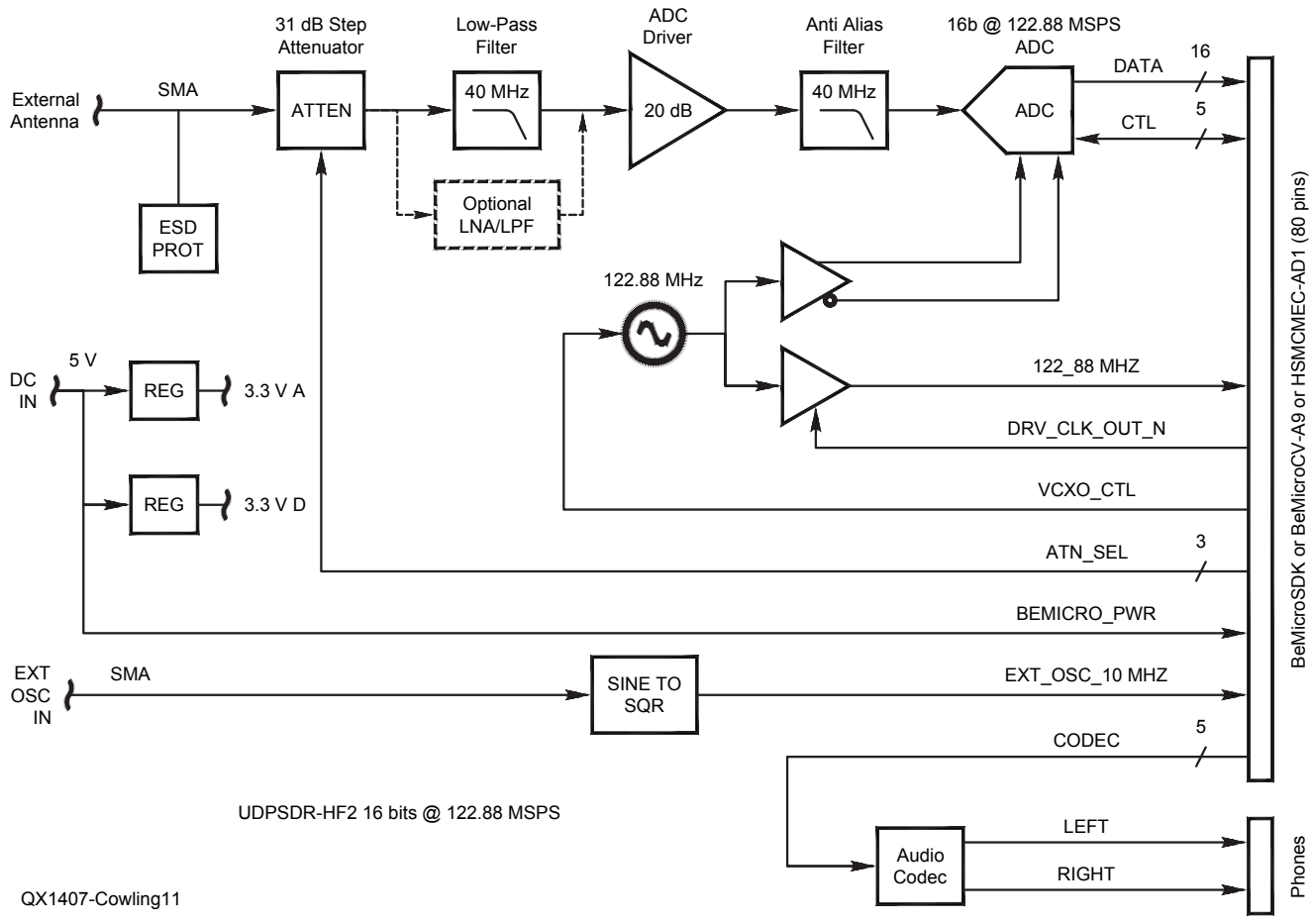


Figure 11 — HF2 receiver block diagram.

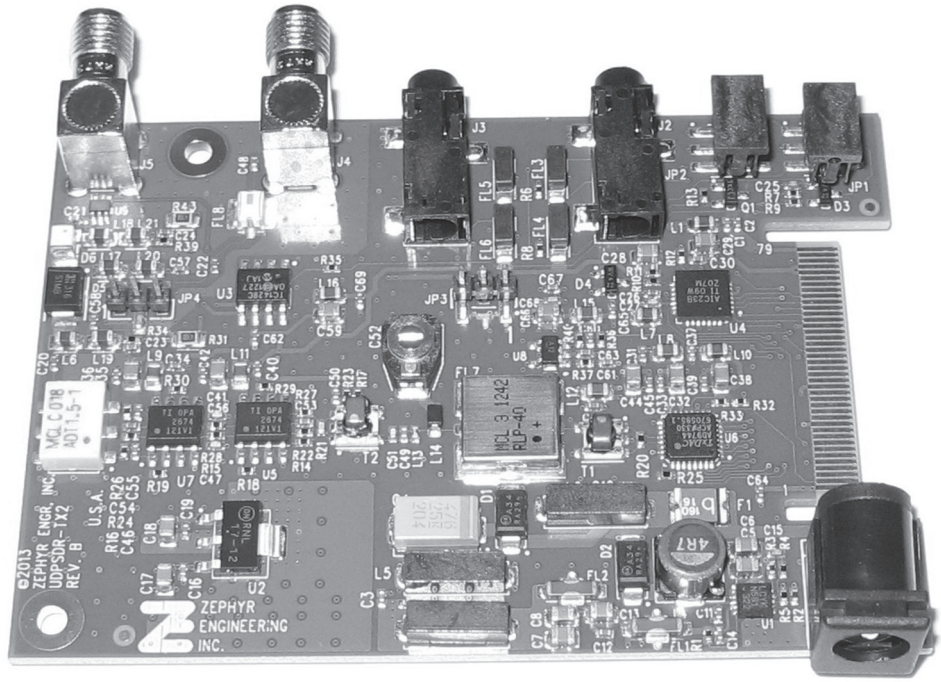


Figure 12 — The TX2 transmitter plugs into either the AD1 adapter or the HF2 receiver.

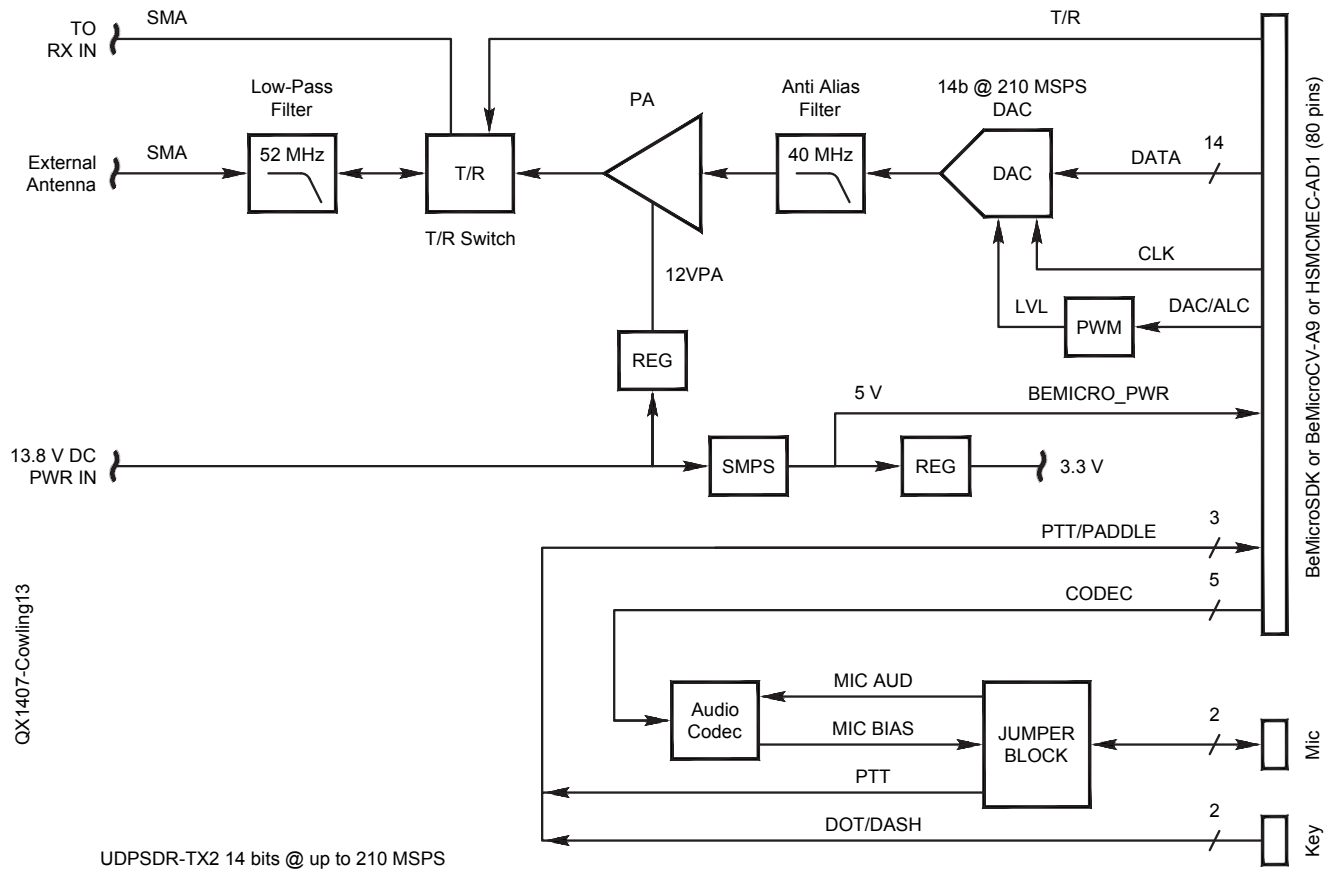


Figure 13 — TX2 transmitter block diagram.

AD9744ARU 14 bit, 210 mega samples per second differential-output-current DAC. A two-stage transformer-coupled differential PA supplies 500 mW to the output connector through a 52 MHz low pass filter and a PIN-diode T/R switch. An on-board +5 V switch mode power supply accepts a wide range DC input (9 V to 18 V) and provides enough current to power an HF2 receiver and either a BeMicroSDK or BeMicroCV-A9 data engine. The TX2 sells for $179 from iQuadLabs.[10]

Amplifiers

The QRPp level of 500 mW may be enough some of the time, but there are other times when you need to trade in your slippers for a pair of boots (speaking in the RF sense, of course). The two options that I describe here are by no means the only viable ones. The first QRO solution is the Hardrock-50 HF power amplifier kit.[11] This kit (Figure 14) is a 5 W input, 50 W output 160 m through 6 m amplifier, but you can add a 10 dB gain, 5 W pre-driver board to make it a perfect match for the 500 mW output of the TX2 transmitter. There is an internal auto tuner in the works, and harmonic filtering is

Figure 14 — The Hardrock-50 HF power amplifier kit covers 160 m through 6 m.

already designed into the amplifier. The TX2 transmitter has an amplifier keying output that can be used to switch the Hardrock-50, or it can be done via RF detection (carrier operated).

TAPR offers the Pennywhistle 18 W power amplifier as a kit and the Alex TX low-pass filter board assembled. Pennywhistle (Figure 15) will produce 16 W to 20 W of power from 500 mW of drive from 160 m through 6 m.[12] Pennywhistle requires harmonic filtering at its output, which can be provided by the Alex TX low-pass filter board (Figure 16).[13] The TX2 amplifier keying output can be used to control Pennywhistle, but there is no control port to control the Alex filter selection. An external manual controller or an interface to a computer or the data engine must be built to perform this function. Both the SoCkit and the BeMicroCV-A9 have extra I/O pins that can be used for this purpose; the BeMicroSDK does not. If you are clever, you can figure out a way to re-purpose unused LED or switch I/O pins on the BeMicroSDK, since you only need two outputs to control Alex-TX.

Systems

Making an SDR

Now that we have discussed all of the pieces, let's assemble them into an SDR. I will examine three SDR configurations, each built from the components described above. To do this, we must focus not on the individual blocks, but on section interconnections, shown as horizontal lines between the three sections shown in Figure 1.

In all three example SDRs, the physical connection between the user interface and the data engine is Ethernet cable. Physical connection is easy, but to get the user interface and the data engine to *understand* each other over this connection requires a bit more effort. To make this work, we will use a standard protocol over the wire called User Datagram Protocol (UDP). A datagram is made up of a header (which contains routing information and a checksum, among other things) and a block of user data. It is up to the originator of each datagram (the source) to place the bytes of user data into the datagram

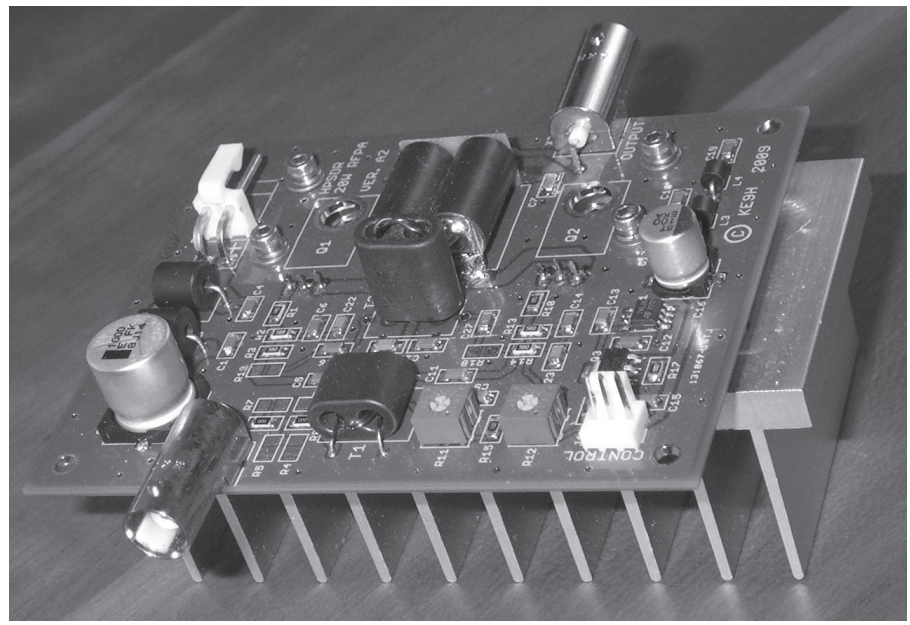


Figure 15 — The Pennywhistle power amplifier requires only 500 mW of drive for 18 W output.

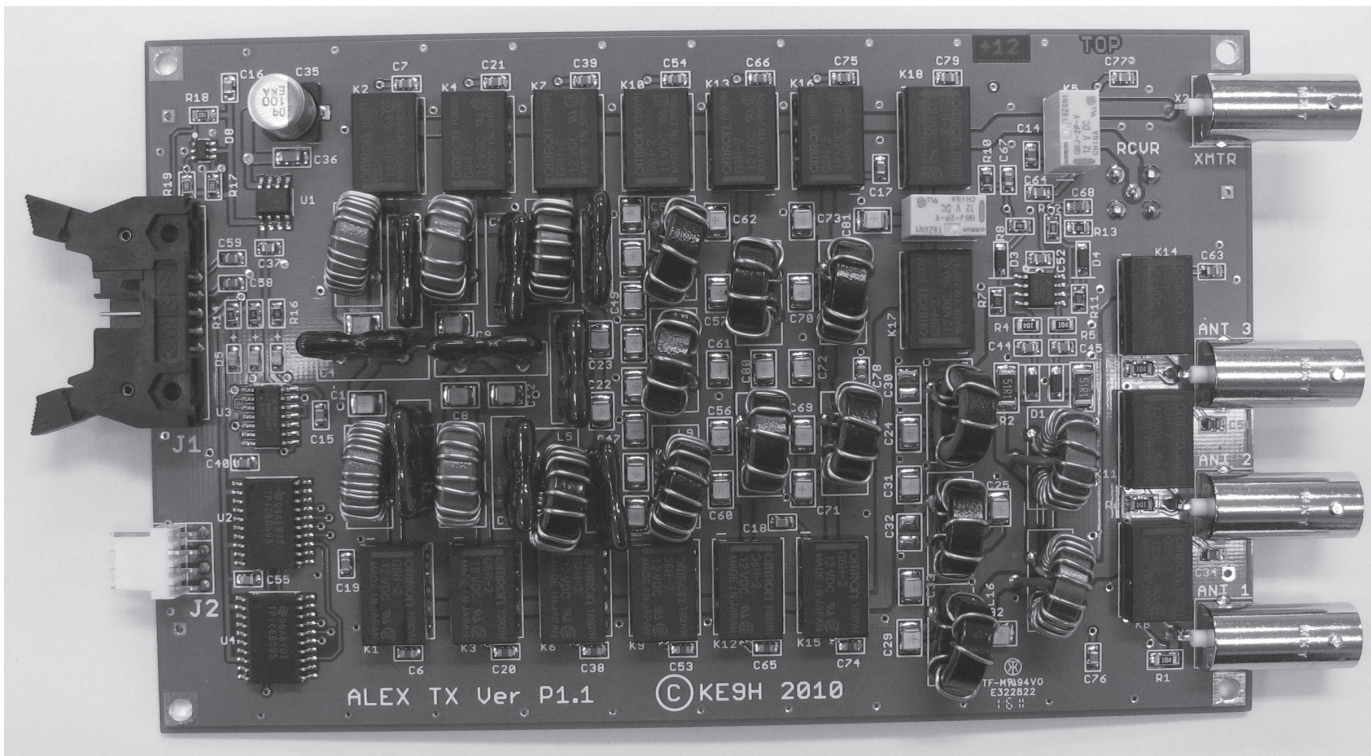


Figure 16 — The Alex transmit low-pass filter board handles harmonic suppression.

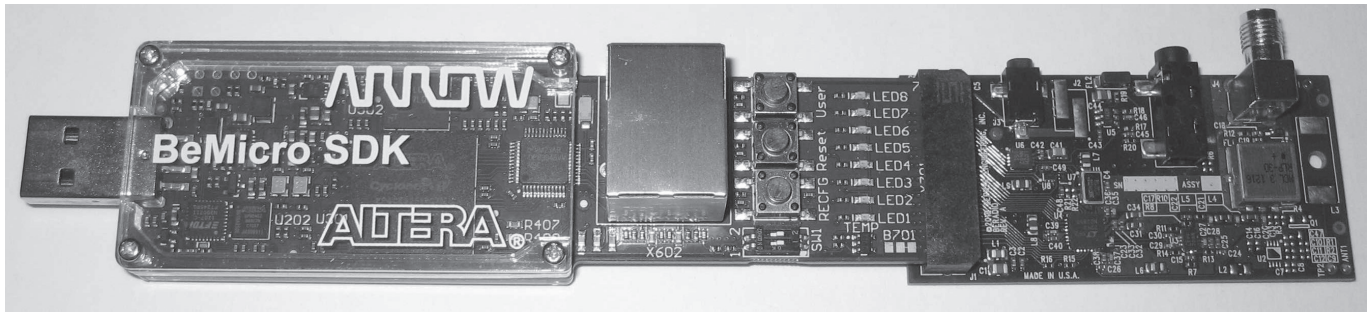


Figure 17 — A complete cost-effective SDR receiver: BeMicroSDK with HF1.

in the order that the receiver of the datagram (the destination) expects. For this to work, the designers of the user interface and the designers of the data engine agree to organize the data within the block of user data in a pre-defined order. Just to confuse things even more, this definition is also called a protocol. For example, the openHPSDR version of PowerSDR™ uses openHPSDR Ethernet Protocol formatted data blocks. The upshot of all of this is that the data engine must build UDP packets that the user interface can understand. There are many different user interfaces and many different data engines, so how can this possibly work? As Paul Simon said, "would you explain about the 50 ways…"[14] Let's look at *three* ways.

Table 1
Comparison of the SDRstick™ HF1 and HF2 Receiver Boards

	HF1	*HF2*
Receive frequency range	100 kHz to 30 MHz	100 kHz to 55 MHz
Antenna connector	SMA	SMA
Front-end attenuator	--	0-31dB, 1dB steps
LPF	RLP-30+ and anti-alias	RLP-40+ and anti-alias
Pre-ADC gain	20 dB	20 dB
ADC	LTC2249, 14-bit at 80 Msps	LTC2208, 16-bit at 122.88 Msps
ADC clock source	TCXO or FPGA	Crystek CVHD-950 VCXO
Receive audio	LTC2641 Audio DAC	TLV320AIC23B CODEC
GPSDO reference input	--	10 MHz
Transmitter expansion micro-edge connector	--	yes
Cost[10]	$169	$399

The ExtIO.dll Method

The High Definition Software Defined Radio (HDSDR) software user interface is a good example of a straightforward way for a single user interface to talk to many different data engines.[15] The designers of HDSDR defined their data block format and wrote their user interface to recognize it. They wrote their code to accept an extension program called ExtIO.dll. This extension sits logically in between their user interface and the data engine and reformats data blocks from the data engine into standard HDSDR format data blocks. Each SDR designer writes a simple ExtIO.dll to convert his data format to HDSDR data format (and back) and includes this software with the SDR. The SDRstick™ radios come with an ExtIO.dll for HDSDR.

Customize Your Data Engine

Another way for one data engine to talk to many different user interfaces is to just program it that way. The "P" in FPGA stands for "programmable," so just program the data engine to format the data blocks for whatever user interface you want to use today. If the FPGA in the data engine has enough resources (like the BeMicroCV-A9 has, for example), program it to recognize the user interface and automatically format the data for that particular program. The SDRstick™ does this, too! When HDSDR or PowerSDR™ user interfaces are started (other user interfaces do this, too), they broadcast a special packet to the network called a Discovery Packet. SDR data engines are designed to reply to this Discovery Packet by returning their network address and radio ID back to the user interface. Since the Discovery Packet also identifies the user interface, (and thus, the data format type that it understands), the data engine can use this information to decide what format to send back. For example, if PowerSDR™ sends a Discovery Packet to an SDRstick™ data engine, SDRstick™ responds with openHPSDR Ethernet format packets. If HDSDR sends a Discovery Packet to an SDRstick™ data engine, SDRstick™ responds with its native format packets, which are then converted to HDSDR native format by the SDRstick™ ExtIO.dll converter.

Customize Your user interface

A third way to hook the data engine and user interface together is to write your own user interface. While this may seem like a difficult solution (and it is not trivial), software can come to the rescue. The GNU Radio project might be the solution that you are looking for. For a GNU Radio receiver, the data engine formats data in its native mode. The SDR designer writes some software called a GNU Radio Source Block that converts the native mode data format into a standard GNU Radio format. Once the data is in this format, it can be used anywhere within GNU Radio to build custom radio software. (In the transmit direction, the interface is called a Sink Block.) GNU Radio has a learning curve to it, but it is an extremely powerful tool with which to build custom SDR user interfaces and applications. SDRstick™ radios come with GNU Radio Source and Sink Blocks. Please refer to "Digital Signal Processing and GNU Radio Companion" by John Petrich, W7FU, and Tom McDermott, N5EG, in this issue of *QEX* for Part 1 of an in-depth look at GNU Radio.[16]

Data Engine to RF Hardware

Connecting the RF front ends to the data engine is simple, since the boards are designed to plug together. A brief description of the interface follows. The HF1/HF2 receiver interface from the ADC is made up of parallel data, an ADC clock and an overflow bit. The TX2 transmitter interface to the DAC is also parallel data and a DAC clock. The CODEC (for receive audio) and the RF step attenuator on the HF1/HF2 receivers and the CODEC (for microphone audio) on the TX2 transmitter are each digitally interfaced to the data engine. The data engine uses several general purpose input/output (GPIO) pins for things like PTT, paddle dot and dash inputs and ADC and clock buffer control. The HF1 (Figure 9),

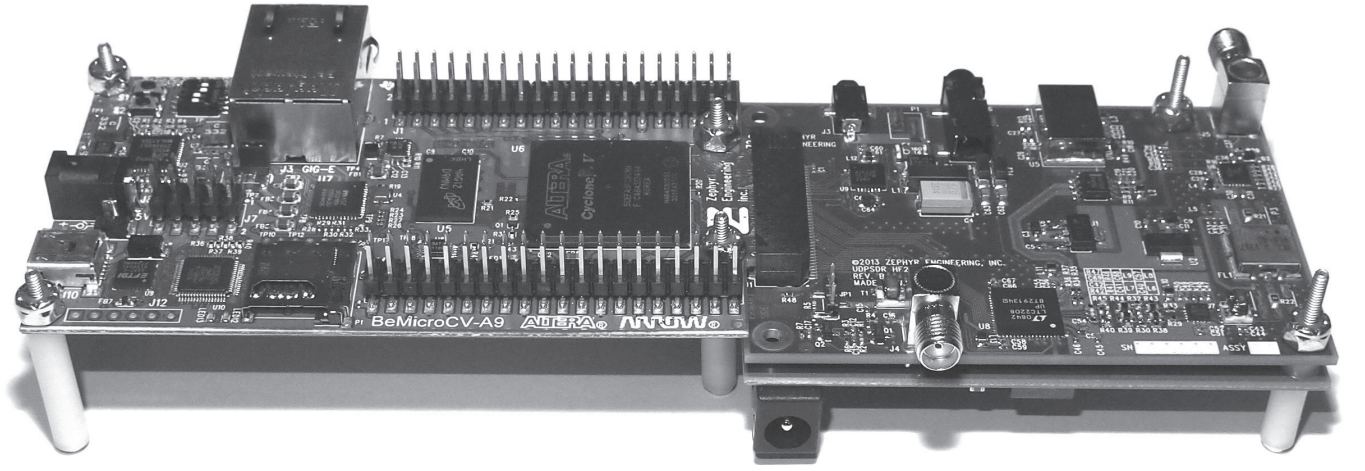


Figure 18 — A complete high-performance SDR transceiver: BeMicroCV-A9 with HF2/TX2.

HF2 (Figure 11) and TX2 (Figure 13) block diagrams show the connections.

Receiver with BeMicroSDK and HF1

The most cost effective SDR platform is the receive-only BeMicroSDK data engine paired with the HF1 receiver (Figure 17). This creates an Ethernet-based receiver that covers 100 kHz to 30 MHz with 1.25 MHz receive bandwidth. This is wide enough to see the entire MW broadcast band on-screen in the panadapter display. It is also wide enough to fully display any of the Amateur Radio bands below 28 MHz, or enough of the 10 m band to display all of the activity on any one mode. This receiver costs just over $200, and is a good place to get your feet wet in FPGA-based SDR. This is by far the least expensive Ethernet-based, broadband SDR available. The FPGA program available for BeMicroSDK/HF1 does native UDP format at 1.25 MHz and 384 kHz receive bandwidths and Hermes UDP format at 384 kHz bandwidth, both receive only.

Transceiver with BeMicroCV-A9, HF2 and TX2

The most exciting SDR platform is the High-performance HF2 receiver married up with the TX2 transmitter and the BeMicroCV-A9 data engine (Figure 18). Truly remarkable SDR functionality becomes possible with the sheer amount of logic present in the A9 data engine's FPGA and the Gigabit Ethernet interface to move data. To put this into perspective, the A9 data engine contains nearly eight times as many logic elements as the Hermes FPGA. While Hermes is limited to about seven virtual receivers, the A9 data engine has no such limitation. (See the "What is a Virtual Receiver?" sidebar.)

The current FPGA program available for BeMicroCV-A9/HF2/TX2 does native UDP format at 1.92 MHz and 384 kHz receive bandwidths, and Hermes UDP format at

MCU Versus FPGA

An embedded microcontroller (MCU) consists of many logic building blocks that are each designed to perform one function. For example, it has an Arithmetic Logic Unit (ALU), which carries out the mathematical operations specified by the instructions in the computer software. It has a Program Counter that keeps track of where in memory the current instruction is located. It has a Memory Management Unit (MMU) that controls accesses to main memory. There are many of these blocks, and each block is a collection of logic gates, memory cells, and transistor switches, each hard-wired to perform one function, and only that function. These small blocks are wired up into a large structure in order to make a functional MCU. This is, of course, a simplistic explanation of how millions of transistors are wired up to form a microcontroller, but it illustrates one main point. The MCU logic and interconnections between these pieces of logic are fixed. The hardware is designed to fetch an instruction and carry it out, fetch the next instruction and carry it out, repeating this process forever. Modern MCUs do this job *very* fast, but they can only perform the operations hard coded into their fixed instruction set. For example, an MCU might have instructions for addition, subtraction or writing to main memory, but it will not have an instruction to perform every complex mathematical operation that might be needed. The programmer writes software to break down these custom, complex mathematical operations into small sequential steps that each can be performed by a pre-defined instruction from the MCU's instruction set.

An FPGA, on the other hand, has very little fixed logic and interconnections. To illustrate this concept, let's imagine that we can take all of the gates and memory elements (small groups of these are called logic elements or LEs) that make up the MCU, disconnect them from each other and spread them out in a "sea of logic elements." If we provide a way to connect these logic elements together in any order we like (in other words, program the FPGA), we can create just about any function we need. In fact, we can connect them back up just the way they were connected in the MCU, and we have (guess what?): an MCU! This is what is called a soft-core processor. One FPGA manufacturer — Altera — has a pre-programmed soft-core processor called NIOS II, but it is not the only one that we can make out of our sea of gates. A soft-core processor is not as efficient as an MCU, since all the logic interconnections take up space on the FPGA chip, making it bigger, and thus, more expensive to make. All the programmable logic interconnects also slow the soft-core processor down because they introduce more delay than the fixed logic interconnects of the MCU.

Soft-core processors are interesting and useful, but they are not the main attraction of FPGAs. Remember that MCUs execute instructions serially? FPGAs can perform their logic functions in parallel. Imagine that I need to perform 10 additions. Even with in-line coding (no loop), it will take the MCU 10 instructions to do this, and more if the 20 addends must be fetched from memory first. If the MCU runs at 100 MHz (10 ns per clock cycle), and we assume that each instruction takes one clock cycle, it will take at least 100 ns to perform the 10 additions. If I program 10 adders into the FPGA, I can perform all 10 additions at the same time, requiring only one clock cycle to obtain all 10 sums. This is a simplistic example, but consider that even small FPGAs have tens of thousands of logic elements, and logic elements number in the millions in large FPGAs. FPGA hardware parallelism creates remarkable capability to implement algorithms that can benefit from this parallelism.

384 kHz receive bandwidth. This only hints at what is possible. For instance, eleven virtual receivers can be designed into the FPGA logic, one for each of the Amateur bands, 160 m through 6 m. The full spectrum of every HF Amateur band (and the lowest VHF one, too!) can be simultaneously displayed. Using appropriate software, eleven users could connect to this radio, and each user would have a virtual receiver. Admittedly, such FPGA code and software does not yet exist. Now that hardware is available to support such features, however, the firmware and software are possible. The BeMicroCV-A9 is a prototype now, but should be available by the time you read this.

Transceiver with SoCkit, AD1, HF2 and TX2

The most flexible SDR platform replaces the BeMicroCV-A9 with the SoCkit development board and the AD1 adapter (Figure 6). The SoCkit board FPGA contains fewer logic elements than the A9 board (110 K versus 301 K), but it has something the A9 does not: a dual-core ARM processor. With this processor (and the other SoCkit on-board resources), we can run an embedded operating system, such as *Linux*. *Linux* brings with it things like a full TCP/IP stack, software control of packet data formatting and easy application development (compared to FPGA applications), among other things. The current FPGA program available for SoCkit/AD1/HF2/TX2 does native UDP format at 1.92 MHz and 384 kHz bandwidths, and Hermes UDP format at 384 kHz bandwidth.

The same virtual receiver scenario is possible with the SoCkit data engine that is possible with the A9, but other possibilities open up with the addition of *Linux* to the system. For example, we could write a server application to serve data up directly to remote clients. We can run this application right on the SoCkit board's local processor, eliminating the computer normally necessary to perform this task. We have made an "NAR," or Network Attached Radio! While I have just coined this term, you can bet that the concept is already here!

Conclusion

Advanced, high-performance hardware is available off-the shelf at reasonable cost. FPGA code is currently available to perform basic functions, while more advanced features are either planned or left to the user to implement. Some open-source FPGA example code is available, and can be used as a starting point for developers and experimenters.[17] There are lots of SDR user interfaces to choose from, many under current development and some are open source.

Notes

[1]Steven W Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, ISBN 0966017633, available for free download at: **dspguide.com**.
[2]Many DSP references can be found here: **dspguru.com/dsp/links/books/online**.
[3]Arrow Electronics BeMicroSDK information: **arrownac.com/solutions/bemicro-sdk**.
[4]Arrow Electronics BeMicroCV information: **parts.arrow.com/item/detail/arrow-development-tools/bemicrocv**.
[5]Arrow Electronics SoCkit board information: **arrownac.com/solutions/sockit**.
[6]Comedians on the TV show *Rowan and Martin's Laugh-In* used the term "sock it to me," typically followed by a dousing with a bucket of water. See **en.wikipedia.org/wiki/Rowan_&_Martin%27s_Laugh-In**.
[7]SoCkit ordering information: **parts.arrow.com/item/search/#st=sockit;renMeR**.
[8]Scotty Cowling. WA2DFI, "The High Performance Software Defined Radio Project," *QEX*, May/June 2014, pp 3-13.
[9]Hermes SDR information: **openhpsdr.org/wiki/index.php?title=HERMES**.
[10]HF1, HF2, TX2 and AD1 boards may be purchased from **iQuadlabs.com**.
[11]Hardrock-50 amplifier information: **hobbypcb.com**.
[12]TAPR Pennywhistle kits: **tapr.org/kits_pw**.
[13]TAPR Alex Filter boards: **tapr.org/kits_alex**.
[14]Paul Simon, 1975: **en.wikipedia.org/wiki/50_Ways_to_Leave_Your_Lover**.
[15]HDSDR web page is: **hdsdr.de**.
[16]John Petrich, W7FU, and Tom McDermott, N5EG, "Digital Signal Processing and GNU Radio Companion: An Easy Way to Include DSP in Your Radio Projects," Part 1, *QEX*, Jul/Aug, Part 2, *QEX*, Sep/Oct 2014.
[17]Hermes FPGA code is open-source: **svn.tapr.org**.

Scotty Cowling, WA2DFI, was first licensed in 1967 as WN2DFI, and has been continuously active since that time. An Extra Class licensee and ARRL Life Member, Scotty is active while mobile on HF CW and on APRS. He is an advisor for Explorer Post 599, a BSA affiliated ham club for teens in the Phoenix, Arizona area. He also enjoys minimalist QRP operating. He has participated in every ARRL Field Day since 1968!

Scotty has been involved in the openHPSDR project for the last 8 years, and has served on the TAPR Board of Directors (2006-2012) and as TAPR Vice President (2011-2012). He is active in the production of openHPSDR components and with other TAPR projects. He is a co-founder of iQuadLabs, LLC, a supplier of openHPSDR systems and other Software Defined Radio components, and is President of Zephyr Engineering, Inc, an engineering consulting firm.

Scotty's professional specialty is FPGA and embedded systems hardware design. He designed his first project with a microprocessor in 1975 and his first FPGA project was in 1987. He holds a BSEE from Rensselaer Polytechnic Institute and an MSEE from Arizona State University.

John Petrich, W7FU
1629 East Lake Sammamish Pkwy NE, Sammamish, WA 98074;
petrich@u.washington.edu

Tom McDermott, N5EG
3950 Southview Ter, Medford, OR 97504;
n5eg@arrl.net

Digital Signal Processing and GNU Radio Companion

The authors present an easy way to include DSP in your SDR radio projects.

Understanding the fundamental identity of the analog and digital signal flow processes makes it easy to apply practical knowledge and hands on Amateur Radio experience to author digital signal processing (DSP) functions using the GNU Radio DSP library with the user friendly graphical interface, GNU Radio Companion (GNU Radio Companion). Does authoring your own DSP programs for your next SDR radio project, interest you? Would you like to develop DSP applications that permit real-time transmit and receive capability as well as simulation ability? Are the thoughts of learning a new programing language and coping with the complexity of DSP intimidating? Would a free and easy to learn DSP software development environment be something you'd want to learn more about? If so, the open source DSP software library, GNU Radio, with the companion graphical user interface, GNU Radio Companion, may be the application for you.

How Complex is DSP?

DSP can be simplified and demystified on a fundamental level by comparing analog and digital signal processing. The signal flow logic of a DSP system parallels the logic of an analog signal flow. Both use similar processing units. Multipliers (mixers), multiply-accumulate (filters), multiplication (amplifiers), and I/O devices, are used in the same flow order to process signals. In the analog realm, a simple, classic analog super heterodyne receiver architecture might involve inputting a signal to an analog mixer, which in turn converts the signal flow to an intermediate frequency (IF). That IF signal is filtered in an analog filter, and then flows to circuits to be detected/mixed to the audio range with output to a speaker. A DSP system signal flow is constructed in the same manner using digitally implemented mixers (multipliers), followed by digitally implemented filters (multiply-accumulate), digitally implemented demodulators, and so on. The DSP signals are rendered in the digital domain and processed at baseband (at or near "0" Hz).

GNU Radio and the Companion Graphical User Interface

With GNU Radio Companion, the user's focus is on rendering the DSP via the graphical overlay rather than having to write code and deal with algorithms and data management issues. With GNU Radio Companion, the user has access to the full functional capabilities of the GNU Radio DSP software library and the ability to author DSP systems. GNU Radio is an open source software library made up of DSP signal processing units written in *Python* and *C++* code.[1] GNU Radio Companion is the graphical overlay on top of the foundational GNU Radio code.[2] With GNU Radio Companion, it is not necessary to learn these foundational codes. The user has access to the full functional capabilities of GNU Radio using only the GNU Radio Companion graphical overlay. A DSP system is comprised of a collection of signal processing units. These units are simply stages in processing a radio signal, rendered in the digital domain. The GNU Radio Companion user connects various signal processing units, rendered as graphical blocks, to author a DSP system. Different blocks and/or arrangements of blocks represent different algorithms and different DSP signal flows. Underneath the GNU Radio Companion overlay, GNU Radio automates the real-time nature of DSP, the handling of buffers, timing, multi-threading, and other software tasks.

[1]Notes appear on page 46.

The Building Blocks of a DSP System Using GNU Radio Companion

The same familiar analog mixers, filters, audio, I/O connections, and so on, rendered in the digital domain, are examples of DSP process units. It is the collection of these process units to form a system that enables the DSP to process information. To author a DSP system, the digital domain signal processing units are sequenced. To build a radio, the sequence of DSP units is connected to an SDR front end and forms the back end of a practical Software Defined Radio.

Within GNU Radio Companion, the foundational signal processing units of GNU Radio are graphically rendered as rectangular boxes with I/O ports, and are called "blocks." There are more than 300 DSP blocks available in the GNU Radio Companion DSP library, which can be used to author DSP projects. These blocks are collected into 46 broad library categories.

A selection of blocks that have recognizable applications for typical Amateur Radio DSP projects are presented in Table 1. The list does not exhaust the range of possible blocks of interest to the creative user. An explanation of Table 1 will help the user to understand the GNU Radio library organization. The first column in Table 1 lists some typical DSP processes, such as a receiver input or GUI controls. The middle column lists the GNU Radio Companion names of the DSP blocks and

Table 1
Selected DSP processes and corresponding GNU Radio Companion blocks

DSP process	*GNU Radio Companion block*	*GNU Radio Companion library category*
Receiver input	FUNcube Dongle, RTL 2832 TV Dongle, Hermes NB SDRstick™, Ettus UHD	FCD, RTL, HPSDR, source UHD
Audio input	Audio source	Audio
Transmitter output	UHD sink, Hermes NB	UHD, HPSDR
Audio output	Audio sink	Audio
Data recording	File sink	Audio
Filtering	Bandpass, Low pass, High pass, Band Reject	Filter
Mod/De-mod	Rx or TX: WBFM, NBFM, AM, PSK, and others	Modulators
GUI display/testing	FFT, Histogram, scope, waterfall, and others	Instrumentation
Networking	TCP, UDP source and sink	Networking Tools
Resampling	Rational and Fractional Resampler	Filters
GUI controls	Radio button, Slider, Tab, and others	Widgets
DSP flow control	Valve, Selector, Null source	Misc., Source

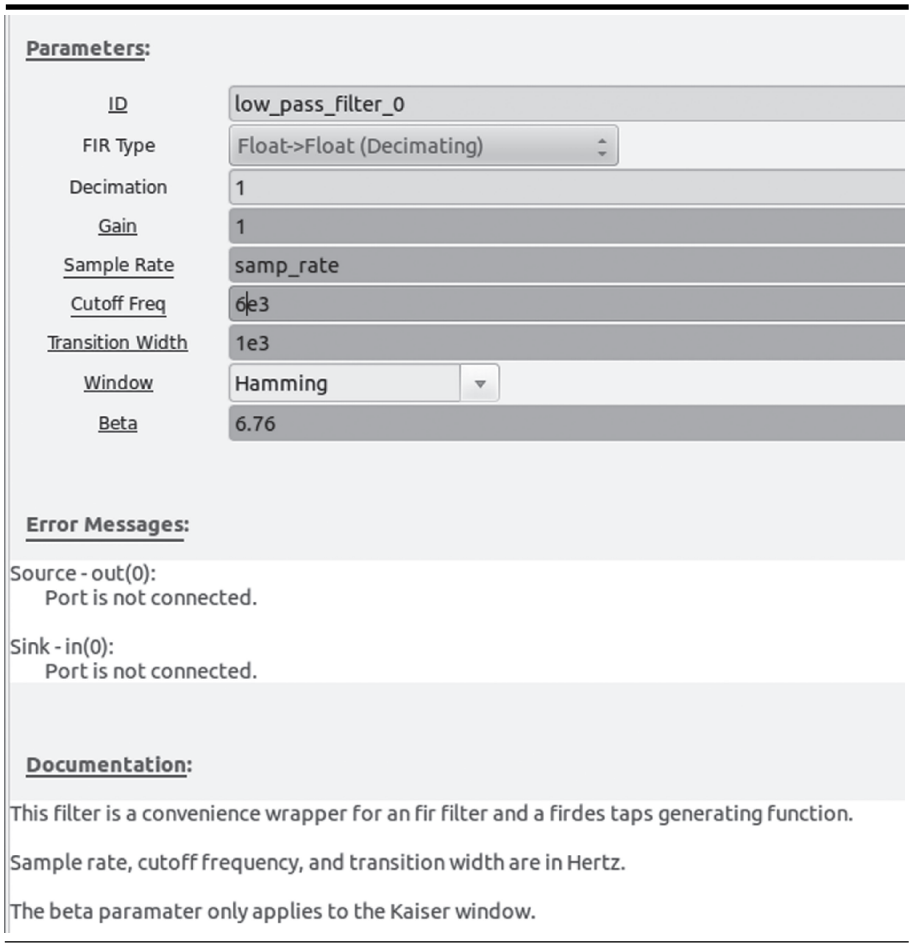


Figure 1 — Low pass filter parameter box.

the right hand column lists the GNU Radio Companion library category in which the particular DSP blocks are collected. The top few rows of Table 1 list the signal sources (such as front ends of SDR receivers) and signal outputs or sinks (such as front ends of SDR transmitters). The list of supported sources and sinks include popular SDR front ends such as the FUNcube Dongle, RTL 2832 TV Dongle, SDRstick™, Hermes/Metis, and the family of Ettus transceivers.[3, 4, 5, 6] Going down the rows:

• The Modulators library category is a collection of standalone, drop in, DSP units for either receiving or transmitting AM, FM, PSK and other modulation modes.

• The Instrumentation library category contains blocks that find dual use as tuning aids in the operational GUI for a radio application and use as trouble shooting tools in the development of DSP projects.

• The Networking Tools category contains blocks to network real world radios with local and wide area networks, a rapidly evolving aspect of radio communications technology.

• The Resampling process is a type of DSP process that permits changing the sample rate of a signal, increasing (interpolating) or decreasing (decimating) the sample rate. These Resampler blocks permit control of sample rates for individual DSP blocks including filters, real world I/O devices, and instrumentation displays.

• Widgets provide the GUI controls for real world operational interfaces. These controls include radio buttons, sliders, and GUI tabs, among others.

• The Miscellaneous category contains selector, valve and null source blocks, which make for practical software control of transmit and receive functions within the SDR transceiver flow graph.

Table 1 illustrates that each signal processing unit performs a limited amount of signal processing and that the blocks must be linked to perform more complex DSP processes. The function of each block is determined by the underlying mathematical algorithm for that DSP process. The algorithm contained in each block is individually programmable. The user programs the DSP block with the mathematical parameters (math variables) that define the desired processing.

An example of user configurable GNU Radio Companion block parameters is depicted in Figure 1, a parameter box for a DSP low pass filter block. The parameter box is brought on screen by double clicking the block of interest. This low pass filter block has been programmed to process a floating point digital connection between blocks. The Cutoff Frequency — bandwidth in this case — is user defined as 6 kHz. The Transition Width — slope of the filter roll off — was chosen to be 1 kHz. Sample Rate, Decimation, and Gain are also user configurable, depending on the particular DSP process being authored.

How Does the GNU Radio Companion User Create DSP Systems in Practice?

The GNU Radio Companion graphical interface makes the authoring of DSP systems easy and approachable. Using GNU Radio Companion terminology, the total DSP signal flow system, graphically rendered by arrow linked blocks, is termed the "flow graph." The flow graph constitutes the totality of the DSP that has been authored. To construct a DSP system using GNU Radio Companion, one moves the desired signal processing block(s) from the on screen library, on the right of the screen, into the work space in the center of the screen. The desired parameters are entered into each block. The ports of all of the signal processing blocks are linked serially with arrow link connections by left clicking the input and output ports of the blocks.

Figure 2 depicts the flow graph of a GNU Radio Companion DSP that generates a 1 kHz audio tone output to the user's computer speaker. A Waveform source block, on the left side of the work area, is connected with an arrow to the Audio sink block (also known as a computer sound card), on the right side of the work area. The user-entered audio frequency parameter is displayed in the Waveform source block as a 1000 Hz cosine. The user selected sample rate parameter for the sound card is displayed in the audio sink block as 48000 Hz. The "Options" box in the upper left corner of Figure 2 identifies the flow graph file name for display purposes and is not a GNU Radio Companion DSP block. The smaller box, labeled as a "Variable" block, is one way to specify the system sample rate as 48000 Hz.

Flow graphs of this type are visually simple and facilitate an intuitive understanding of the DSP system function. The math variables that define the process, entered as parameters, are displayed within each block. Trouble shooting in a GNU Radio Companion flow graph is as easy as following the signal flow logic illustrated by the arrows, and by visualizing whether the parameters displayed in each block are correct. GNU Radio Companion provides automatic error messages if ports are not properly connected and if parameters are not entered correctly. GNU Radio Companion also contains a category of instrument tool blocks such as FFT, oscilloscope, constellation and auto correlation displays that can be used to troubleshoot and optimize the DSP flow graph. The combination of the GNU Radio Companion visualization of the signal processing blocks, the automatic error messages and the included instrument blocks facilitate the rapid layout and validation of your DSP flow graph.

Step By Step Guide to Get GNU Radio With GNU Radio Companion Up and Running

GNU Radio is a continuously evolving open source DSP software library with GNU Radio Companion (GRC) as the graphical user interface. The entire DSP library and graphical interface package is designated by the GRC version number. Installation of a GNU Radio Companion version automatically installs the GNU Radio DSP software library that supports the GNU Radio Companion interface.

To get started with GNU Radio Companion, there are four decisions:

1) The computer hardware platform,

2) The operating system (OS),

3) The GNU Radio Companion installation method,

4) Deciding which version of GNU Radio Companion to install

Computer Hardware Requirements

Consider your intended use of GNU Radio Companion to decide how much compute capability you need. Basic learning with simple DSP systems is easy with inexpensive computer systems such as a single-core, 2 GB computers, and are used successfully by the authors as starting points. These computers are fine for learning and demonstration projects, though the process of down loading and installation of GNU Radio Companion versions with this level of computer can take several hours.

If your goals are to develop more elaborate DSP systems where real time performance is important, and to easily keep up with GNU Radio Companion version updates, a high performance computer is a wise choice. Dual and quad core CPUs with clock speeds above 2.5 GHz, and 4 GB or more of RAM are preferred for more elaborate DSP development, such as transceiver DSPs. High performance transceivers implemented in GNU Radio Companion typically consume less than 2 GB of memory, and involve no page file swaps. With this level of computer the installation of the GNU Radio Companion package can be completed in less than an hour.

Choosing A Computer Operating System (OS)

Linux Ubuntu is the preferred OS, versions 12.04 and above, with the most recent Ubuntu release 14.04 LTS preferred for new installations. It is a small step to learn *Linux* Ubuntu and implement GNU Radio Companion in a *Linux* environment. GNU Radio Companion can be installed on *Linux* Ubuntu, *Windows* OS and *Mac* OS, but non-*Linux* installs seem plagued by problems, judging from Forum comments, though some *Windows* and *Mac* users do claim success.[7] Experienced *Windows* and *Mac* OS users

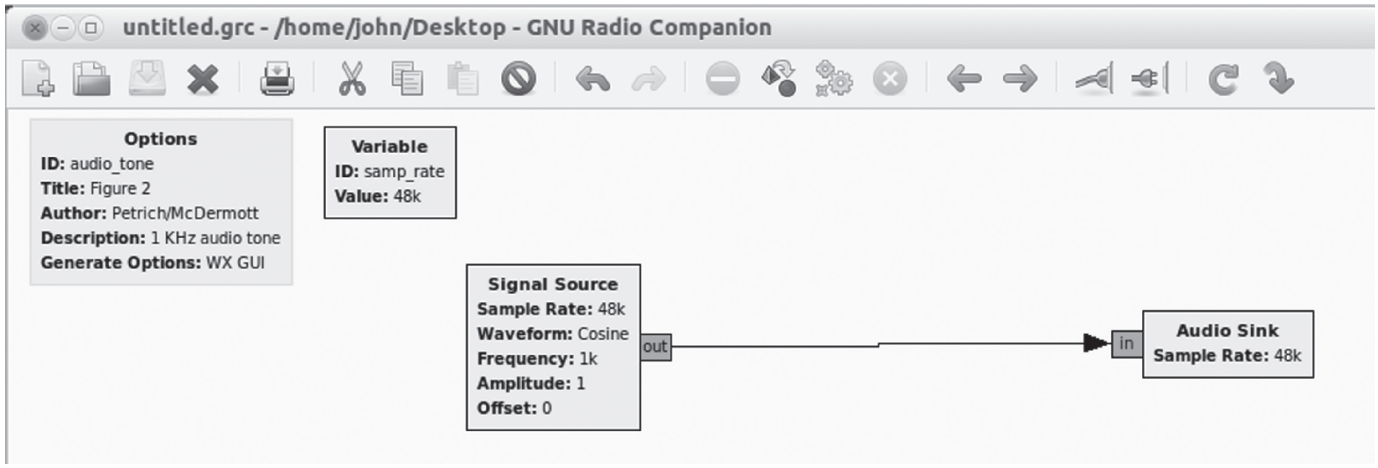


Figure 2 — GNU Radio Companion DSP Flow Graph: The signal source is a 1 kHz audio tone, which is connected to the audio sink (output) to a computer speaker.

may have good results at managing the GNU Radio Companion installation process, while less skilled users are probably wise to take the path of least resistance and choose the *Linux* Ubuntu OS.

Some users are hesitant to learn a new operating system. These concerns are greater than they need be if the user allows for an initial period of adjustment and occasional reference to Ubuntu help resources.[8] The application installation in *Linux* is quite simple and easily mastered. The installation of GNU Radio Companion requires some basic familiarity with a handful of simple commands in the *Linux* Ubuntu terminal application (see Note 8). In the end, *Linux* Ubuntu has the look and feel of a late model *Windows* operating system. Experienced *Windows* users find *Linux* Ubuntu as logical, business like, and as easy to understand as *Windows*. *Linux* offers the same right click functionality as *Windows*, which greatly simplifies file manipulation. "Right click is your friend" in *Linux*.

Choosing a GNU Radio Installation Method

The third decision involves how to install the *Linux* Ubuntu based GNU Radio Companion application. Fortunately, several possibilities exist. Table 1 shows some common options. "Boot and go" is a great approach for those with limited computer skills, or for those who want to quickly install GNU Radio Companion and experiment. Bootable USB 2.0 memory sticks and DVD discs are available and provide both the Ubuntu operating system and the GNU Radio Companion application in one simple package. The "boot and go" approach, allows the user to run GNU Radio Companion in the *Linux* Ubuntu operating system and avoid making any_changes to the host computer hard drive. The "live" GNU Radio Companion / *Linux* bootable media limit you to certain versions of Ubuntu and GNU Radio Companion.

To "boot and go", you insert the media into the appropriate USB port or DVD drive, and boot into *Linux* Ubuntu from that media. Once booted into *Linux*, the user can execute GNU Radio Companion or use various available Ubuntu applications such as an e-mail client or Internet browser. Shutting down the computer closes the *Linux* operating system. On restart, the computer will automatically default to the native operating system, unless commanded by the user to again boot from the media into *Linux*. A downside is the reduction in performance due to the limited speed of the USB or DVD data bus. As a consequence, the Ubuntu boot process and the execution of any applications are somewhat slower than what one expects of the same process(s) using a standard hard drive (HD). With "boot and go," the user cannot save DSP files. Despite these limits, the "boot and go" method makes it very easy to quickly get started with GNU Radio Companion and delay decisions regarding alternative GNU Radio Companion installation approaches.

Alternatively, Ubuntu *Linux* and GNU Radio Companion can be installed directly on the user's computer hard drive. A hard drive installation is very efficient, and uses the computer's native processing performance. A dual boot installation offers the flexibility of changing the operating system as needed. There are numerous resources, on-line and text books that will help with this process. The text by Helmke and Graner (see Note 8) is a very helpful resource and is recommended for those new to *Linux* Ubuntu. The book is reasonably priced ($23 paperback, $15 Kindle), explains the *Linux* Ubuntu operating system, and the command line terminal application. The paperback book also includes an installation CD of the latest Ubuntu 14.04 LTS operating system software, along with a helpful hard drive partition utility. With that book, and others like it, a trouble free dual boot installation of *Linux* Ubuntu and *Windows* operating systems is easily accomplished. A free on-line source of Ubuntu 14.04 LTS installation software is available and works well.[9]

Choosing a GNU Radio Companion Version

GNU Radio is automatically embedded with GNU Radio Companion when one installs any GNU Radio Companion version. A GNU Radio Companion version choice is a decision for those whose goals are to fully exploit the potential of GNU Radio Companion and prefer the GNU Radio Companion installation to occur to their hard drive —a choice not applicable to the "boot and go" approach.

Typical of open source software applications, GNU Radio and GNU Radio Companion are in a state of continual evolution. The current version 3.7.3 at the time of this writing has become quite stable. Blocks written for older versions of GNU Radio (3.6 and earlier) may not function in 3.7, but most of the older blocks have now been updated for 3.7 compatibility.

For a beginning user installing the latest version GNU Radio Companion, 3.7.3 is the best choice. Installing the earlier stable version, GNU Radio Companion 3.6.5.1, allows the user to easily exploit the reservoir of previously published GNU Radio Companion DSP projects that can be found on the Internet. All GNU Radio Companion versions are available via free downloads. Table 2 shows how to find and install the latest and legacy versions of GNU Radio Companion.

Adding DSP Blocks to the GRC DSP Library

GNU Radio Companion versions, even the latest version GRC 3.7.3 doesn't include all of the blocks available for the GNU Radio DSP software library. The DSP library of GNU Radio can be updated without changing GNU Radio Companion versions. Of particular interest to Amateur Radio operators, all GNU Radio Companion versions natively include many common radio interface blocks in the GNU Radio DSP library. The radio interface blocks

Table2:
Methods to install Linux Ubuntu and GNU Radio Companion

Install Method	*Boot from:*	*Where to find:*	*Comments:*
"Boot and Go"	DVD and USB versions. Can boot on large range of computer hardware. Allows user to use a Windows or Mac computer to experiment in Linux without altering the hard drive.	**DVD: http://gnuradio.org/redmine/news/36 USB: https://www.ettus.com/product/details/LIVEUSB**	Slow operation due to reduced media data rates. Can only save designs to the boot media.
Native *Linux* and GNU Radio (GRC)	Install *Linux* and GNU Radio (GRC) to hard drive	***Linux* Ubuntu: www.ubuntu.com/download/desktop/install-ubuntu-desktop** **GNU Radio Companion: http://sbrac.org/files/build-gnuradio**	Requires learning at least a little Linux. (See text)

are used to link actual SDR hardware into GNU Radio DSP, to build a complete SDR. Updates can include radio interface DSP blocks not natively installed in older GNU Radio Companion versions or new radio interface blocks as new SDR front ends and interfaces become available. In GNU Radio language, these uninstalled blocks are referred to as "Out of Tree" (OOT) blocks. The term "OOT blocks" refers to blocks not automatically included in the GNU Radio DSP software library. Installation details are typically published with the blocks. Examples of new radio interface blocks are the TV Dongle, Hermes NB, the SDRstick blocks, and Whitebox, which interface GNU Radio with the TV Dongle, HPSDR Hermes/Metis, SDRstick™, and Whitebox SDR front ends, respectively.[10, 11, 12] Other DSP blocks and package add-ons to the GNU Radio library can be updated as well.

To install new blocks in GNU Radio Companion, first locate and download the block. Installation is completed with a short series of commands in the *Linux* terminal window to install the block into the GNU Radio library. Locating a new block is as simple as locating the block repository on the Internet. Table 3 lists some repositories for additional blocks.

Step-By-Step Procedure for Installing Ubuntu *Linux*

The following procedure is one method to install Ubuntu 14.04 LTS on a PC. Note that this may irretrievably replace the existing operating system.

1) Download and un-compress Ubuntu 14.04 LTS .iso to the desktop from this site:
www.ubuntu.com/download/desktop/install-ubuntu-desktop.

2) Create a boot media by following the instructions on the website.

3) Boot your computer from the boot media. Access the boot menu on most PCs by selecting F-12 at the splash screen.

4) Select "Install Ubuntu" and follow the on-screen directions to complete the installation. The boot medium contains a user friendly partition utility. Selecting 30 to 50 GB of hard drive space for Ubuntu is sufficient.

5) After the installation is complete, remove the boot media and configure the operating system following the on-screen directions.

Step-By-Step Procedure for Installing GNU Radio

The following procedure is recommended as one reliable method to install GRC3.7.3 and above into Ubuntu 14.04. The procedure consists of five steps:

1) This step (step 1) is not necessary if you have not previously attempted to install GNU Radio Companion onto your hard drive. Otherwise you will need to remove prior GNU Radio Companion installations and residuals, including "distribution" versions downloaded via the Ubuntu Software Center. Even if you have "removed" GNU Radio Companion via the Software Center, follow this procedure. Copy and paste the "Gnuradio_remove" script file into your Ubuntu Home directory. **http://svn.tapr.org/filedetails.php?repname=OpenHPSDR+Main&path=%2Ftrunk%2FN5EG%2FGRC3.7%2FGnuradio_remove**

2) Copy and paste the latest available "build-gnuradio" script file provided by Marcus Leech (**http://sbrac.org/files/build-gnuradio**) into the Ubuntu Home directory.

3) Make sure that GNU Compiler Collection is installed.

4) Execute the "build-gnuradio" script file.

5) Open the GNU Radio Companion application and check that the application is functional.

Step 1:

(Note: $ is the prompt character displayed by the terminal, don't type it yourself.) Open the Terminal and at the "~$" prompt enter:

$ sudo apt-get purge gnuradio (and Enter)

Step 2:

1) Copy and paste the file "Gnuradio_remove" to your Home directory as a "shell script."

2) Open your Home directory in the graphical browser.

3) Right click the file "Gnuradio_remove" and select "Properties". A popup will open.

4) Select the "Permissions" tab.

5) Select the check-mark "Allow executing file as a program"

6) Close the popup menu then close the Home directory.

7) Open the Terminal. At the "~$" prompt enter

~$ sudo ./Gnuradio_remove (and Enter) (This command instructs the operating system to execute the remove script located in the Home directory.)

Step 3:

Determine if GCC (GNU Compiler Collection) is installed on your system. Using the Terminal, at the "~$" prompt enter:

~$ which gcc (and Enter)

If the result is a directory listing such as: /usr/bin/gcc then GCC is installed and no further action is necessary.

If the result is a blank line, then GCC is *not* installed, and it is necessary to install it for GNU Radio Companion to build. The package "build-essential" includes the compilers, linkers, make, and cmake. In the Terminal, at the "~$" prompt enter:

~$ sudo apt-get install build-essential or the alternative command:

$ sudo apt-get --reinstall install build-essential (and Enter)

Step 4:

Execute the "build-gnuradio" script from your Home directory.

At the Terminal "~$" prompt enter:

~$ ./build-gnuradio –m (and Enter) (This command instructs the operating system to execute the build script located in the Home directory and get the latest version of GNU Radio Companion.)

(Note: The build process can take a long time depending on the capabilities of your computer. Computers at the i7 level require about 20 minutes to complete the build process. Computers at the i3 level can take several hours to complete the build and single core computers longer yet. A number of "y"

Table3
Installing GNU Radio Companion Versions From Source

Recommended Installation Steps	*Source*	*Command(s)*
Download the Build GNU Radio (GRC) script, and execute it. Can take 1-8 hours depending on computer.	**http://sbrac.org/files/build-gnuradio**	Download it using web browser.
Install GNU Radio (GRC) 3.7.3 (latest version) (recommended)		Use *Linux* Terminal command: **./build-gnuradio –m**
Command to install GNU Radio (GRC)3.6.5.1 (Older version that is compatible with legacy published DSP flow graphs.)		Use *Linux* Terminal command: **./build-gnuradio –o**

Table4
Sources for Adding Selected "Out of Tree Modules" (Blocks) to the DSP Library

Out of Tree DSP Block	*Location*
Hermes / Metis HPSDR interface.	**http://svn.tapr.org/repos_sdr_hpsdr/trunk/N5EG**
SDRstick™ interface	**http://svn.sdrstick.com/listing.php?repname=sdrstick-release**

commands, for "yes", are required in the Terminal during the initial stages of the build process, so pay attention.)

Step 5:

To open GNU Radio Companion after the build is compete, open the Terminal, and at "~$" enter:

~$ gnuradio-companion (Enter)

Step-by-Step Procedure for Installing Out-Of-Tree Modules into GNU Radio

To install additional DSP block types, not included in the official released version of GNU Radio (called 'out-of-tree') there are three steps:

1) Download and uncompress the source code file for the desired block into a directory (below the Home directory). In this example we will refer to the gr-hpsdr Hermes NB block. Your home directory is usually identified as ~ and we want to uncompress the complete structure into a subdirectory of ~. That complete new block will include files and even deeper directories (for example, build). Normally the uncompress program takes care of setting it all up for us. For example, you might create a new directory ~/gr-hpsdr and unzip the code into it. In *Linux*, directory names can include the dot character.

2) Enter a series of commands in the Terminal that will compile and install the source code into the GNU Radio DSP library. The commands are those that follow the "$" sign in this text, for example:

~/ gr-hpsdr/build/$ make

Open GNU Radio Companion and confirm the presence of the block(s) in the library.

Step 1:

Create a new directory and check to make sure that the uncompressed block files are present in the Home directory.

In the Terminal at "~$"enter:

~$ mkdir gr-hpsdr (Enter)

~$ cd gr-hpsdr (Enter)

~/gr-hpsdr$

Use the graphical archive manager to extract the source code into the newly created directory.

~/gr-hpsdr$ ls (Enter)

A list of the files in your new directory will be listed, including the source directory for the block you are intending to install, in this example the directory: gr-hpsdr. It should include some subdirectories, such as apps, build, cmake, python, and others, and the file CmakeLists.txt. Check to see if you have the build subdirectory, if not create it.

~/gr-hpsdr$ mkdir build (Enter)

Step 2:

Open a Terminal, and at the "~$" prompt enter the following series of commands:

~$ cd ~/ gr-hpsdr/build (Enter)

~/ gr-hpsdr/build/$ cmake ../ (Enter)

[Note the space after "cmake" and before "../"]

~/ gr-hpsdr/build/$ make clean (Enter)

~/ gr-hpsdr/build/$ make (Enter)

~/ gr-hpsdr/build/$ sudo make install (Enter)

~/ gr-hpsdr/build/$ sudo ldconfig (Enter)

~/ gr-hpsdr/build/$ exit (Enter) and the terminal will close.

Step 3:

Open the GNU Radio Companion application and verify that the desired block is installed in the DSP library list. If you installed gr-hpsdr, for example, there would now be an hpsdr selection on the panel with all the gnuradio blocks, and inside that hpsdr selection will be the hermesNB block. Other out-of-tree modules will have other names of course.

Congratulations. You now have a working installation of GNU Radio Companion on your computer. In Part 2 of this article we will present several examples of what you can do with GNU Radio.

ARRL member, Official Observer, and Amateur Extra class licensee John Petrich, W7FU, was first licensed as K6OJV in 1955 and then as W7HQJ after moving to Seattle. He is a practicing physician, and Clinical Associate Professor of Psychiatry, School of Medicine, University of Washington. John is active in community affairs, enjoys family life, sea kayaking and cycling. John's radio experience parallels the evolution of radio communications technology over the past century. He started with a homebrew crystal receiver followed by a much loved and modified single tube regenerative receiver in a cardboard box. Upon earning his license, he graduated to operating QRP using a crystal controlled 6V6 tube transmitter constructed on a wooden chassis. Subsequently his rigs evolved from modified surplus gear to home constructed to full featured analog receivers and transmitters. John's first love is CW, the prototypical digital QRP mode. He credits radio with endless opportunities for engaging learning opportunities and long lasting friendships. At present John's rig is an experimenter's station built around the HPSDR Atlas bus system. The station is supplemented with back-up rigs using Ettus and SDRstick™ SDR "front ends" and DSP "back ends" built using GNU Radio Companion. John is interested in communicating with others who have similar interests.

ARRL Life Member, and Amateur Extra class licensee Tom McDermott, N5EG, has been licensed 45 years. He is a member of TAPR, IEEE, and Internet2, and has been involved in the development of fiber optic transmission and switching systems since the initial deployment of single-mode fiber in positions ranging from ASIC designer to CTO. He currently is a participant in the IEEE 802.3 Ethernet 100GE and 400GE standards projects. Tom has a BSEE degree from the University of California, Berkeley, and has written one textbook on wireless digital communications. He's been involved in many computer-related ham projects, from the TEXNET layer 3 packet radio system, to a VNA project, network simulation, and other efforts. His current interest is using a HPSDR Hermes SDR transceiver and GNU Radio to experiment with DSP algorithms.

Notes

[1]The GNU Radio Wikipedia page provides more detailed information about this software package: **http://gnuradio.org/redmine/projects/gnuradio/wiki**

[2]The home page for GNU Radio Companion is found at: **http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion**

[3]For details about the Fun Cube Dongle, go to: **www.funcubedongle.com/**

[4]There is more information about the SDRstick at: **http://sdrstick.com/**

[5]The TAPR website has detailed information about the High Performance Software Defined Radio project, including the Atlas backplane and the various cards that plug into the backplane to create the radio: **www.tapr.org/**

[6]To learn more about Ettus Research and their universal software research peripheral (USRP) hardware visit the Ettus website: **www.ettus.com/**

[7]You can find a lot of information and answers to common questions on the GNU Radio forum at: **https://www.ruby-forum.com/forum/gnuradio**

[8]Matthew Helmke and Amber Graner, *The Official Ubuntu Book*, Seventh Edition, Prentice Hall, ISBN -13: 978-0-13-301760-1.

[9]You can find on-line instructions and files to install Ubuntu at: **www.ubuntu.com/download/desktop/install-ubuntu-desktop**

[10]There is more information about the Hermes and Metis hardware on the TAPR website: **www.tapr.org/kits_hermes.html and www.tapr.org/kits_metis.html**

[11]For information about the SDRStick hardware, see: **http://sdrstick.com/**

[12]More information about the novel Whitebox SDR project is available at: **radio.testa.co/index.html#document-faq**

Upcoming Conferences

Central States VHF Society

July 25 – 27, 2014
Austin Marriott South
4415 South IH35
Austin, TX 78744
Hotel Reservation Phone
888-253-1628

The 48th Annual CSVHFS Conference will be held July 25–27, 2014. There will be a *Conference Proceedings*, which will be available at the conference.

Presentations and Posters at the conference may be technical or non-technical but will cover the full breadth of amateur weak signal VHF/UHF activities. The presentations generally vary from 15 to 45 minutes, covering the highlights with details in the *Proceedings* paper. Topics of Interest include:

- VHF/UHF Antennas, including modeling/design, arrays and control
- Construction of Equipment – such as transmitters, receivers and transverters
- RF power amplifiers – including single and multi-band, vacuum tube and solid state
- Preamplifiers (low noise)
- Regulatory topics
- Software defined radio (SDR)
- Test equipment – including homebrew, using and making measurements
- Operating — including Contesting, Roving and DXpeditions
- Propagation – including ducting, sporadic E, tropospheric and meteor scatter
- Digital Modes – WSJT, JT65 and others
- EME (Moon Bounce)
- Digital Signal Processing (DSP)

Banquet Speaker

The Saturday evening Banquet Speaker will be Jimmy Treybig, W6JKV.

Registration information and other details are available at the Society website: **www.csvhfs.org/2014conference**.

The 33rd Annual ARRL and TAPR Digital Communications Conference

September 5-7, 2014
Austin Marriott South
4415 South IH-35
Austin, TX 78704
Hotel Reservation Phone:
512-441-7900

Now is the time to start making plans to attend the premier technical conference of the year, the 33rd Annual ARRL and TAPR Digital Communications Conference. This year's DCC will be held September 5 – 7, 2014 in Austin Texas, at the Austin Marriott South. This is the same hotel as the Central States VHF Society Conference. Regular attendees will note that the conference is a couple of weeks earlier than normal this year. It is the weekend after Labor Day.

The ARRL and TAPR Digital Communications Conference is an international forum for radio amateurs to meet, publish their work, and present new ideas and techniques. Presenters and attendees will have the opportunity to exchange ideas and learn about recent hardware and software advances, theories, experimental results, and practical applications.

Topics include, but are not limited to: Software defined radio (SDR), digital voice (D-Star, P25, WinDRM, FDMDV, G4GUO), digital satellite communications, Global Position System (GPS), precision timing, Automatic Packet Reporting System® (APRS), short messaging (a mode of APRS), Digital Signal Processing (DSP), HF digital modes, Internet interoperability with Amateur Radio networks, spread spectrum, IEEE 802.11 and other Part 15 license-exempt systems adaptable for Amateur Radio, using TCP/IP networking over Amateur Radio, mesh and peer to peer wireless networking, emergency and Homeland Defense backup digital communications, using Linux in Amateur Radio, updates on AX.25 and other wireless networking protocols and any topics that advance the Amateur Radio art.

This is a three-Day Conference (Friday, Saturday, and Sunday). Technical sessions will be presented all day Friday and Saturday. In addition there will be introductory sessions on various topics on Saturday.

Join others at the conference for a Friday evening social get together. A Saturday evening banquet features an invited speaker and concludes with award presentations and prize drawings.

The ever-popular Sunday Seminar has not been finalized yet, but is sure to be an excellent program. This is an in-depth four-hour presentation, where attendees learn from the experts. Check the TAPR website for more information: **www.tapr.org**.

Call for Papers

Technical papers are solicited for presentation and publication in the *Digital Communications Conference Proceedings*. Annual conference proceedings are published by the ARRL. Presentation at the conference is not required for publication. Submission of papers are due by 15 July 2014 and should be submitted to: Maty Weinberg, ARRL, 225 Main Street, Newington, CT 06111, or via the Internet to **maty@arrl.org**. There are full details and specifications about how to format and submit your paper for publication on the TAPR website.

Even if you are not presenting a paper at the conference, plan to bring a project or two to display and talk about in the popular Demonstration Room, or "Play Room" as it is commonly known.

AMSAT Symposium

October 10-12, 2014
Double Tree by Hilton Baltimore
— BWI Airport
890 Elkridge Landing Rd
Linthicum, MD 21090
Hotel Reservation Phone:
410-859-8400

The 2014 AMSAT Symposium and General Membership Meeting will be held in the Baltimore/Washington DC area, at the Double Tree by Hilton Baltimore — BWI Airport, on the weekend of October 10-12 2014.

Whether you are a seasoned satellite operator or think you might like to give it a try, the Symposium will have plenty of presentations of interest. Catch up on the latest news about AMSAT's various satellite projects as well as presentations about equipment and operating practices. The Call For Papers has not been posted to the AMSAT website at the time this column was being prepared, but now is the time to start thinking about any paper you might be interested in writing, for presentation at the Symposium, or for inclusion in the *Proceedings* book.

Check the AMSAT website (**ww3.amsat.org)** for updated information. Mark those dates on your calendar now, and plan to attend.

Microwave Update

October 24-25, 2014
Rochester Marriott Airport
1890 Ridge Road West
Rochester, NY 14615
Hotel Reservation Phone: 585-248-8640 or 800-228-9290

The Rochester VHF Group (RVHFG) is hosting MUD 2014 at the Rochester Marriott Airport Hotel. Microwave Update is an annual technical conference and includes presentations by leading Amateur Radio microwave experimenters. Attendees from all over the world have the opportunity to discuss the latest technical developments and operating achievements taking place in the 902 MHz-and-up amateur radio frequencies. There will be test equipment for measurements, including noise figure up to 47 GHz.

The conference will conclude with the Saturday evening dinner banquet. For early arrivals on Thursday, there will be a tour at the nearby Antique Wireless Association (AWA) Museum.

Microwave Update 2014 — Call For Papers

The Microwave Update 2014 program committee is calling for papers and presentations on the technical and operational aspects of microwave Amateur

Radio communications. Papers will be published in the conference proceedings (print and CD). Many will also be selected for presentation at the conference.

The deadline for proceedings paper submissions is August 15, 2014. The Microsoft Word file format (text) is preferred for these papers. The deadline for the presentation version of selected papers is September 1, 2014; Microsoft PowerPoint (slides) file format is preferred for presentations.

Even if you will be doing a presentation at the conference, please try to make your proceedings paper submission more than just the outline slides from the presentation. We would like the published proceedings to provide full content for people who are not able to attend the conference presentations.

Detailed formatting information for authors (margins, photos, other files) is provided on the website.

Please e-mail your papers, as well as questions or comments regarding the technical program, to Bill Rogers K2TER: **k2ter@rochester.rr.com**.

Solicited topic areas include:

- Centimeter, millimeter, submillimeter and light wavelengths.
- Antenna design, simulation, construction, measurement, application.
- Microwave building blocks (LNAs, PAs, LO chains, Mixers, Synthesizers, Filters, and so on).
- Transverters (single and multiband).
- Fixed station, Rover and Beacon design, packaging and operation.
- Operating techniques, software and other aids.
- Weak signal propagation modes and enhancements.
- New or unusual emission modes (ATV, digital modulation, wide area packet networks, and similar topics).
- Practical effects and limits of phase noise, antennas, path characteristics on various emission modes.
- Microwave components (affordable and available modern commercial components; homebrewed; surplus).
- Repeaters (microwave bands and/or unusual modes like ATV, packet WAN).
- Construction techniques (SMT, wirebond, microstrip, waveguide, substrates, homebrew).
- Measurement equipment and techniques (tuning amplifiers or filters, optimizing noise figure, measuring phase noise, antenna patterns and gain; professional results on homebrew/shoestring budgets).
- CAD (preferably free or low cost) for circuit, antenna, path and system simulation and design.
- Conversion of surplus microwave equipment.
- Or — suggest your own topics.

Submissions may range from short notes to full length technical papers, original research to hints and tips, new designs to surplus conversions, professionally engineered to hacked on a shoestring budget.

Survey papers that summarize current know-how and tutorials that help and encourage newcomers are also welcome. Some topics may be organized and presented as workshops (for example, construction and measurement techniques).

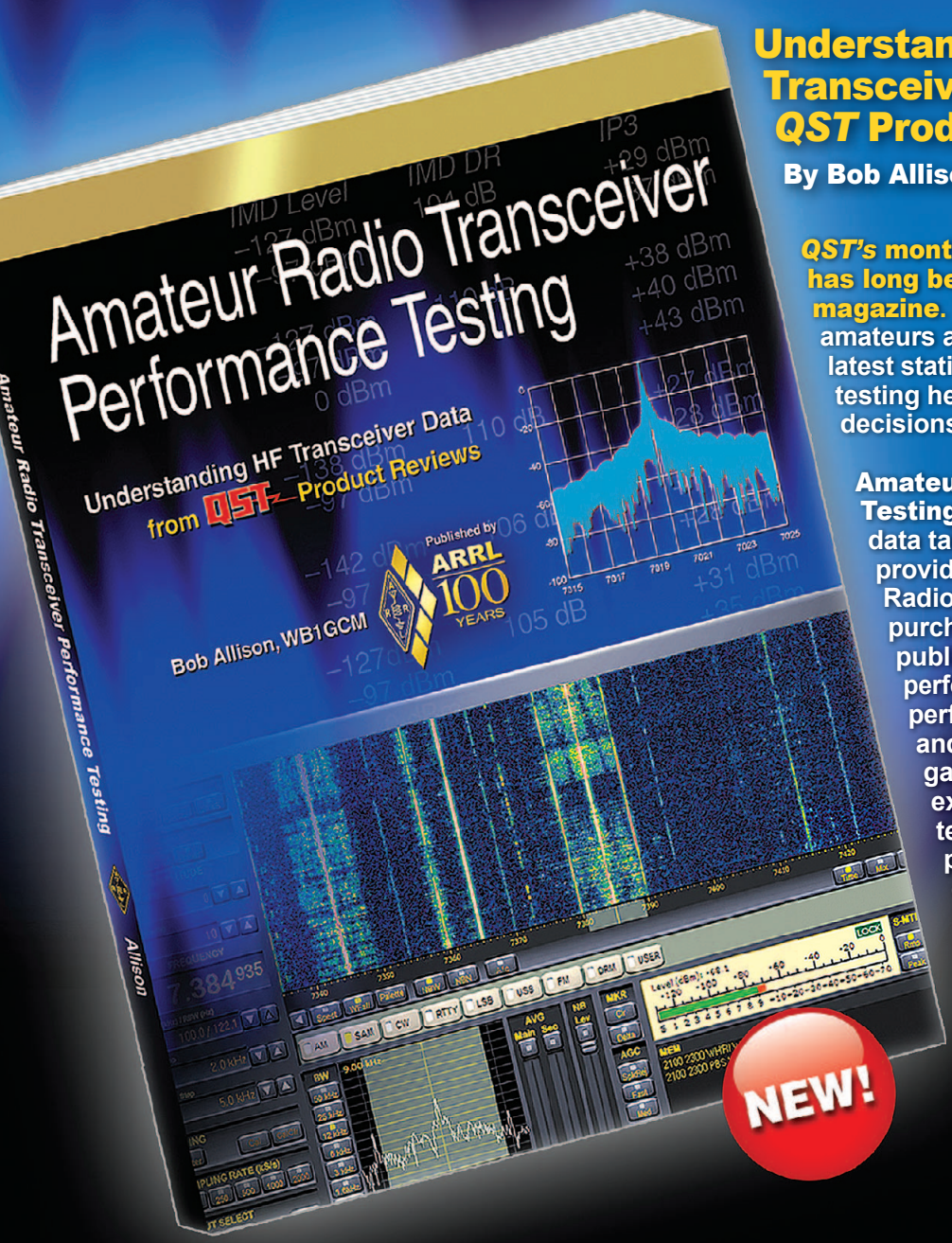
We are looking forward to seeing you and your presentation at MUD 2014.

Bill Rogers K2TER: **k2ter@rochester.rr.com**.

MUD 2014 Technical Program Chairman

For further details and registration, go to the Microwave Update 2014 website **www.microwaveupdate.org**.

Amateur Radio Transceiver Performance Testing



Understanding HF Transceiver Data from *QST* Product Reviews

By Bob Allison, WB1GCM

***QST's* monthly "Product Review" column has long been the most-read section of the magazine.** That's not surprising as most radio amateurs are interested in reading about the latest station equipment — and product review testing helps operators make informed decisions based on their needs.

Amateur Radio Transceiver Performance Testing explains in detail the performance data tables from *QST* Product Reviews, providing a valuable resource for Amateur Radio operators who are looking to purchase a transceiver. It discusses how published laboratory data relates to actual performance, how each major test is performed, the significance of each test, and what the numbers mean. You'll gain a better understanding of the extensive testing ARRL performs, technical terms and parameters presented in Product Review, and develop the capability to reach your own conclusion about which HF transceiver is best for you.

NEW!

Amateur Radio Transceiver Performance Testing

ARRL Order No. 0086

Special ARRL Member Price!

Only **$19.95*** (retail $22.95)

*plus shipping and handling

ARRL The national association for AMATEUR RADIO®

ARRL 100 YEARS

www.arrl.org/shop

Toll-Free US 888-277-5289,
or elsewhere +1-860-594-0355

QEX 7/2014

Quicksilver Radio

See Our Huge Display at ARRL National Centennial Convention 2014. Visit Our Booths 504-505-506-507-508-509-510-511

Componet Analyzers

LCR Inductor, Capacitor, Resistor Analyzer

An advanced instrument that greatly simplifies the testing of passive components. Traditional LCR bridges are inherently complex and very time consuming to use. This does everything automatically, it tells you the component type in addition to component value data. What's more, it automatically selects the best signal level and frequency for the component under test.
Just clip the universal test leads to your component and press the test button. In seconds, analyzer will identify the type of component together with the component's main value. Further component data is also displayed, such as the DC resistance of an inductor. The test frequency is automatically selected to suit the component under test and this is also confirmed on the scrollable display.

Automatic component and pinout identification, connect the test leads any way.
Gain measurement for transistors.
Gate threshold for Enhancement MOSFETs.
Depletion mode detection for MOSFETs.
Forward voltage drop measurements of diodes, LEDs and transistor base-emitter junctions.
Base-Emitter shunt resistor detection.
Collector-Emitter diode detection; Intelligent diode network descriptions; Short circuit detection between any 2 or 3 junctions.
Bipolar transistors; Diode protected transistors (C-E); Resistor shunted transistors (B-E); Darlingtons; Diodes; Diode Networks; Enhancement mode MOSFETs; Depletion mode MOSFETs; Junction FETs (JFETs); Sensitive Thyristors; Sensitive Triacs; LEDs, Bicolor LEDs

DCA Discrete Component Analyzer

DCA-Pro Semiconductor Analyzer

Components supported include:
-Transistors (including Darlingtons), Silicon and Germanium types. Measures gain, Vbe and leakage.
-MOSFETs, enhancement mode and depletion mode types. Measure on-threshold (at 5mA) and approx transconductance (for span of 3mA-5mA).
-JFETs, including normally off SiC types. Measures pinch-off voltage (at 1uA) and approx transconductance (for span of 3mA-5mA).
-IGBTs (insulated gate bipolar transistors). Measures on-threshold (at 5mA).
-Diodes and Diode networks, LEDs, bicolor LEDs (2 and 3 lead types).
-Zener Diodes with measurement of zener voltage up to 9V at 5mA.
-Voltage regulators (measures regulation voltage, drop-out voltage, quiescent current).
-Triacs and Thyristors that require less than 10mA of gate current and holding current
The instrument can be used stand-alone or connected to a PC. Either way, the DCA Pro will automatically identify the component type, identify the pinout and also measure a range of component parameters such as transistor gain, leakage, MOSFET and IGBT threshold voltages, pn characteristics and much more.
When connected to a PC using the supplied USB cable, a range of low current curve-tracing functions can be performed. Various graph types are available, with more to follow:
-Bipolar transistor output characteristics, Ic vs Vce.
-Bipolar transistor input characteristics, Vbe vs Ib.
-MOSFET transfer function, Id vs Vgs.
-JFET transfer function, Id vs Vgs.
-PN junction I/V curves, forward and reverse options (for Zener diodes).
Curve tracing is performed using test parameters in the range of +/-12V, +/-12mA. All curve-tracing data can be instantly pasted into Excel© for further graphing and analysis. PC Software is included with the DCA Pro on a Peak USB memory stick. Software designed for Windows XP or Windows 7.

Quicksilver Radio Products
Sign up on our website for your **free** newsletter. Ham Radio news, articles and special discounts.

www.qsradio.com