Ham 97J – ASV Control – Pi software COS

Dr. Marc 231012

RE: AllStar nodes using Raspberry Pi, software COS and RPT_ETXKEYED

Traditional AllStar nodes (and most digital ham processes) require a URI (USB Radio Interface) in additional to the Raspberry Pi which is doing the heavy lifting. Since the Pi has all the computing power and interface pins, why not use it without a URI? Rather employ just a simple, cheap, unmodified sound card and a few jumpers.

To complete the task, we need access to a single variable! So, we are looking to the computer knowledgeable ham community for help with that last link. It's the way of open source.

I. The process is much easier than it first appears. Rather than skip to the end, let's build to that point.

1. As you are likely aware, Asterisk is more of a toolbox than a pick-it-up-and-run program. The shell is there, and it is very good, but the items to do a task, such as build a node, must be assembled.
2. The directories shown in this write-up are on the node, using the Pi with Asterisk app_rpt installed. These specifics are using HamVoip, but are similar on AllStarLink. *WinSCP* or similar is a good tool to see the structure.
3. When setting up a node, the user selects an input/output channel, such a SimpleUsb or SimpleRadio.
4. The channel is described by a module in
   /usr/local/hamvoip-asterisk-2022-01-22/lib/asterisk/modules/.
5. A list of these modules which are loaded at start up is in.
   /etc/asterisk/modules.conf
6. Check to make sure your module is included has been a part of initial node setup instructions.
7. SimpleUsb channel module is chan_simpleusb.so.
8. During setting up the node, parameters such as levels and inverted signals are set and saved in
   /etc/asterisk/simpleusb.conf.
9. The radio operating parameters are saved in
   /etc/asterik/rpt.conf
10. One of the rpt.conf variables defines the channel:
    rxchannel=SimpleUSB/usb
11. Some of the available channel drivers are explained in
    https://wiki.allstarlink.org/wiki/Rpt.conf

Value	Description
dahdi/pseudo	No radio, used for hubs
SimpleUSB/usb_1999	SimpleUSB (limited DSP)
Radio/usb_1999	USBRadio (full DSP)
voter/1990	RTCM
Pi/1	Raspberry Pi PiTA

12. Rpt.conf has multiple stanzas within its list of variables.
    [events] control activities when a state changes.
    [functions] control activities on DTMF or a command.

II. Now consider how the interface described in the /module/channel must work.

1. Audio in and Audio out are transferred from real world to the Pi. The present architecture is built on a USB sound card using a CM119/108 DSP chip set.
2. URIs use the same chip set, but make a physical connection for PTT and COS, adding ~$100 to the process.
3. The DSP pins are connected to real devices on one side and translate to a software input for Asterisk.

```
;gpio1=in                ;default COS input
;gpio4=out1              ;original uri
;gpio8=out0              ;COS LED
```

4. Asterisk takes the software pin state and uses it to set variables as defined in the simple_usb module channel.
5. Asterisk is a multi-tasking package, whose module channel sets, clears, and tests a suite of globally accessible showvars such as RPT_TXKEYED, RPT_RXKEYED, RPT_ETXKEYED.

6. For example, in rpt.conf [events] stanza the following line sets the COS LED when receiving.
```
cop,62,GPIO4:1 = c|t|RPT_RXKEYED
```
7. Another example in [events] gives the time when transmitter is keyed.
```
/usr/local/sbin/saytime.pl 582202  = s|t|RPT_TXKEYED
```
8. In other words, the parallel nature of Asterisk allows using and setting the showvars as explained in.
   https://wiki.allstarlink.org/wiki/Event_Management.

III. Use the Pi in lieu of a URI board.

1. The AllStarLink site shows there is channel driver for Raspberry Pi PiTA. It is set in rpt.conf by rxchannel= Pi/1. See the table above.
2. As of now, I have not found that driver or any info.

IV. So, with the normal open-source, Linux, do-it-yourself approach, this is the state-of-affairs.

1. The audio remains the same as the simple_usb. The USB sound card uses the same CM119 DSP chipset.
2. The PTT, TX LED, and COS LED are outputs from the Pi and are set-up precisely like #6 with the RPT_TXKEYED triggering the defined Pi pin.
3. *The kicker is getting the COS line variable into rpt.conf.*
4. As seen above in simple_usb channel, the traditional COS variable comes from 'gpio.1' on the sound card DSP.
5. In deciphering the chan_simpleusb.so module, a statement declares this port cannot be changed. So, it is set someplace. If we can find access to that variable, the project is about complete.

V. State of the Pi conversion.

1. BASH scripts to start and kill the Pi process are written and operating.
2. All scripts are saved at
   /etc/asterisk/local
3. Python script to take care of interrupt from a mic button push is complete, tested, and operating. The results are available on a discreet variable and on RPT_ETXKEYED.
4. Python scripts to turn on/off LEDs and PTT are written and tested.
5. All this is packaged and script loaded at start-up on the next line after Asterisk load.
6. It is well integrated as just another package within Asterisk.
7. No major rewrites of Asterisk main code is necessary. Simply the integration of another package is required.
8. The COS variable from the Pi / Python script is available at Asterisk CLI and Linux CLI when the mic button is pressed. Notice Asterisk listing.
```
piblue*CLI> rpt showvars 582202
Variable listing for node 582202:
RPT_ETXKEYED=1
XX_RPT_ETXKEYED=0
RPT_TXKEYED=0
```

VI. What remains is still eluding.

1. How to equate the rpt.conf and simple_usb.conf 'gpio1=in' variable to the Pi / Python GPIO, either RPT_ETXKEYED or an alternate.
2. That appears to be it. (But we know other stuff always arises.)

VII. When we get this app completed, it will greatly simplify building a node and reduce the cost by about half. This is a real, open-source solution.

If you have any feedback, ideas, or know someone who may have legit suggestions, that would be greatly appreciated.