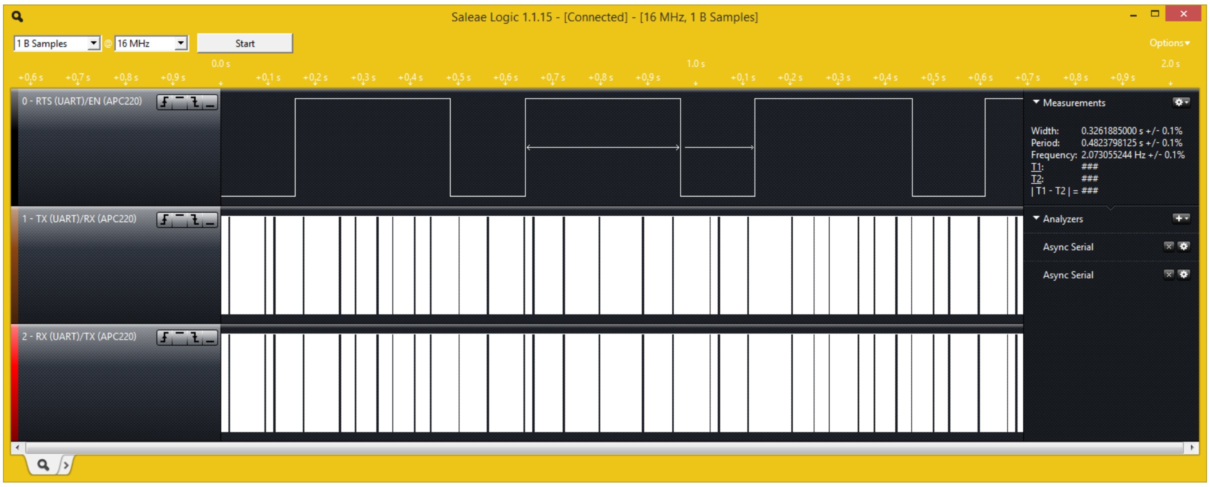
Analysis of APC220 RF Magic

No APC200 Radio Connected

With no APC220 radio connected to the UART and RF Magic running, we get this:-

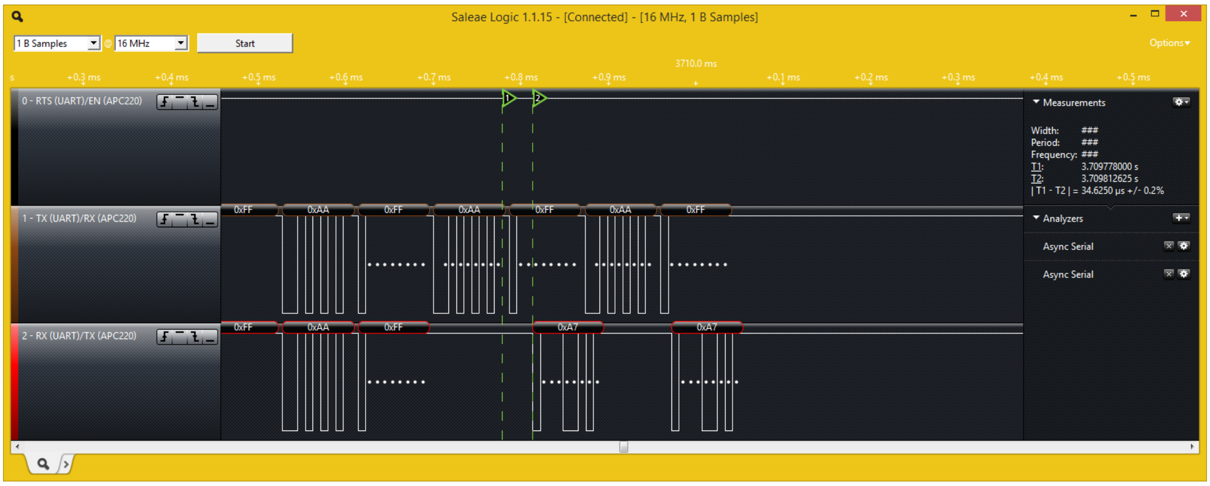
- Sending 0xAA 0xFF at 115,200 baud full line rate
- The signal on the UART RX channel is just crosstalk
- RTS pin on UART alternating 326ms on/156ms off:-



APC220 Radio Connected and Detected by RF Magic

- As soon as APC220 radio is connected and detected, RTS stays high

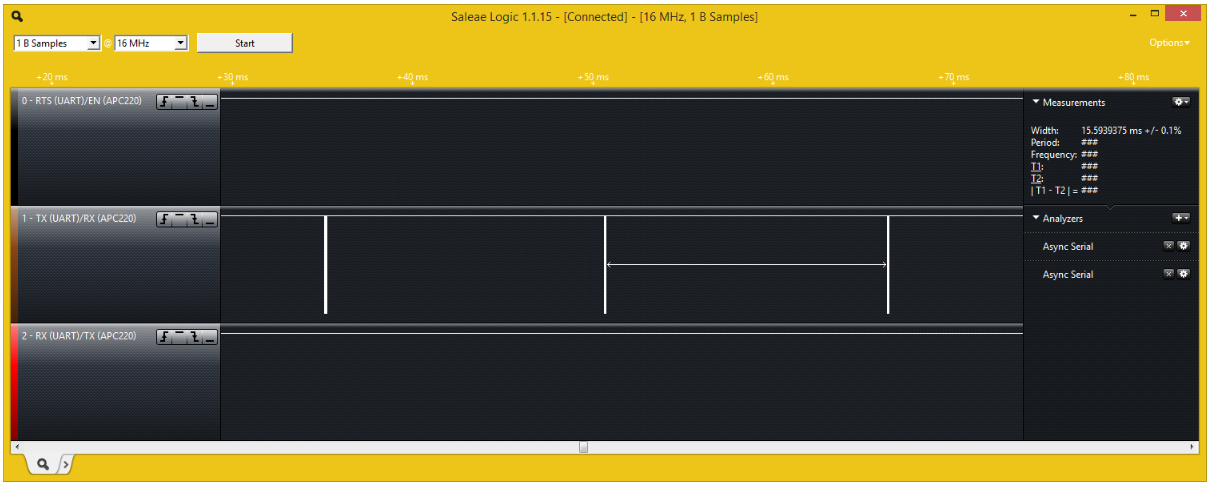
- APC220 starts responding to 0xAA with 0xA7



- It looks like the transmitted 0xFF in the detection phase might not be important. This is good, since it seems to overlap with the incoming 0xA7.

- For detection to work, the 0xA7 "ping" from RF Magic seems to be required to come at some minimum frequency. On a fast PC running the loop at full speed, there is a gap of 2.95ms and detection works reliably.

- On a slow PC with a gap of 15.59ms, detection does not work:-



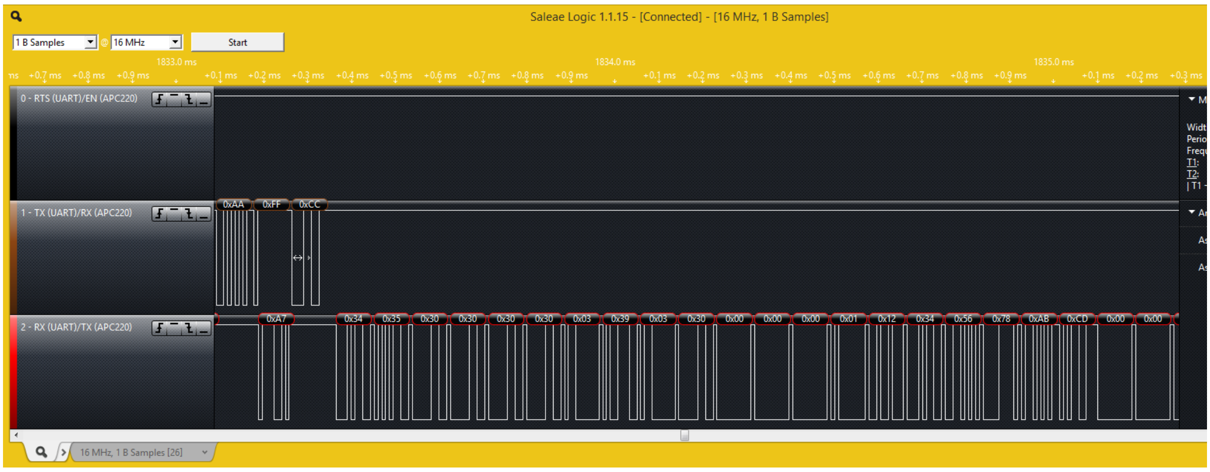
Commands

Read

To read the current radio settings, send a single byte of 0xCC. The APC220 will respond with a block of 32 bytes as follows:-

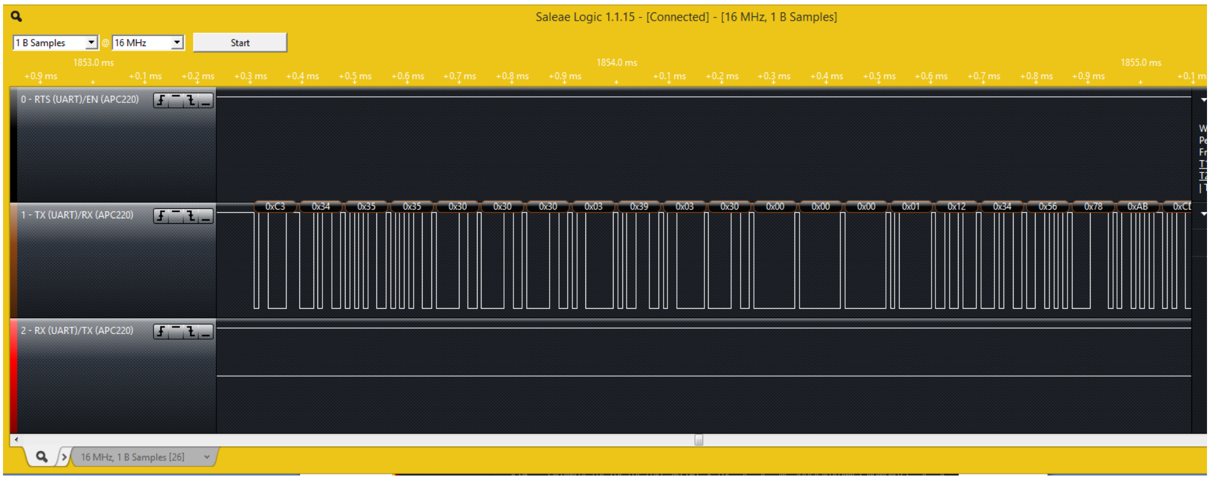
<table>
<tr><td>0</td><td rowspan="6">Frequency in KHz, ASCII-encoded</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td><td>Radio baud rate (sometimes an ASCII value, sometimes a binary value ☹; 0=1200, 1=2400, 2=4800, 3=9600, 4=19200)</td></tr>
<tr><td>7</td><td>RF power (ASCII value, 0…9)</td></tr>
<tr><td>8</td><td>Serial baud rate (ASCII value, 0=1200, 1=2400, 2=4800, 3=9600, 4=19200, 5=38400, 6=57600)</td></tr>
<tr><td>9</td><td>Serial parity (ASCII value, 0=none, 1=odd, 2=even)</td></tr>
<tr><td>10</td><td>Always 0</td></tr>
<tr><td>11</td><td>Always 0</td></tr>
<tr><td>12</td><td rowspan="2">Network ID (16 bits, 12=MSB, 13=LSB)</td></tr>
<tr><td>13</td></tr>
<tr><td>14</td><td rowspan="5">Node ID (MSB -> LSB)</td></tr>
<tr><td>15</td></tr>
<tr><td>16</td></tr>
<tr><td>17</td></tr>
<tr><td>18</td></tr>
</table>

19	
20	Always 0
21	Always 0
22	Always 0
23	Always 0
24	Always 0
25	Always 0
26	Always 0
27	Always 0
28	Always 0
29	Always 0
30	Read=0xNN, Write=0x00. 0xNN appears to be radio-specific (i.e. varies from one radio to the next). Writing to it has no effect.
31	Always 0xD2



Write

To update radio settings, send the character 0xC3 followed by the same block of 32 bytes described above. Byte #30 should be set to 0x00. It will read back as 0x9A.



Notes

- The fact that the first 10 bytes of the response to a Read command are ASCII (i.e. high nybble=3) and indeed the very first byte is always 0x34 is useful: it can be used to reliably detect the start of a the command response if it is not possible to reliably flush the serial buffer after issuing the Read command (0xCC)

The EN pin

- The EN pin on the APC220 **must** be pulled high during normal operation. If it is low, it will neither transmit nor receive data.
- Problem is that most terminal clients will leave it low if RTS/CTS flow-control is disabled. Need to either
    - Separate the EN pin out and pull it high
    - Use custom communications software that allows RTS (which is the pin on the UART that connects to EN) to be set high
- When it is low, the APC220 is in "programming mode" which allows AT-type commands to be sent to it to program it (less capable than RF Magic, though)

Python version of RF Magic

- Requires Python version >= 2.7
- Requires pyserial module
    - "easy_install pyserial" or "pip pyserial", depending on platform

Read

```
C:\CanSat\rfmagic>python rfmagic.py read com3
Detected APC220 radio.  Reading current settings...
```

```
Read():
Frequency         = 418.000
Radio Baud Rate  = 9600
RF Power          = 9
Serial Baud Rate = 9600
Serial Parity     = None

done!
```

```
C:\CanSat\rfmagic>python rfmagic.py read -v com3
Detected APC220 radio.  Reading current settings...
Read():
Frequency         = 450.000
Radio Baud Rate  = 2400
RF Power          = 9
Serial Baud Rate = 2400
Serial Parity     = None
Network ID        = 0x0001
Node ID           = 0x12345678abcd
Radio ID (fixed) = 0x9a

done!
```

```
C:\CanSat\rfmagic>python rfmagic.py write -h
usage: rfmagic.py write [-h]
                        writeserialport frequency {0,1,2,3,4,5,6,7,8,9}
                        {1200,2400,4800,9600,19200}

positional arguments:
  writeserialport       Serial port
  frequency             Frequency (NNN.N : 418.0 - 455.0 in 200KHz steps)
  {0,1,2,3,4,5,6,7,8,9}
                        RF power
  {1200,2400,4800,9600,19200}
                        Baud rate (line and radio)

optional arguments:
  -h, --help            show this help message and exit
```

```
C:\CanSat\rfmagic>python rfmagic.py write com3 450.0 9 2400

Detected APC220 radio.  Reading current settings...
Read():
Frequency         = 418.000
Radio Baud Rate  = 9600
RF Power          = 9
Serial Baud Rate = 9600
Serial Parity     = None

Writing new settings to radio...these are the values returned during the "write"
 process:
Write(): f=450.000MHz, radio baud=2400, power=9, serial baud=2400, parity=None
Frequency         = 450.000
Radio Baud Rate  = 2400
```

```
RF Power         = 9
Serial Baud Rate = 2400
Serial Parity    = None

Reading (hopefully) updated settings from radio...
Read():
Frequency        = 450.000
Radio Baud Rate  = 2400
RF Power         = 9
Serial Baud Rate = 2400
Serial Parity    = None

done!
```