Running MFC/R2 with Sangoma Wanpipe® using OpenR2 (libopenr2)

December 2, 2008

Prepared By
Jim Van Meggelen
Core Telecom Innovations Inc.

Executive Summary

MFC/R2

The MFC/R2 protocol and its variants are in use all over the world[1]. This protocol remains popular because it is generally less expensive than ISDN-PRI, yet offers many of the advantages that PRI has over analogue.

> MFC/R2 stands for MultiFrequency Compelled/Region 2. Commonly called R2, this protocol is over 50 years old, and used to be a very common signalling mechanism throughout most of the international public telephone networks. It has now been mostly replaced by SS7 and ISDN, however it is still common in many parts of the world.

The biggest challenge in implementing R2 is that there is no single standard that is adhered to throughout the world. In each country where the telecom carriers offer R2 circuits, the actual mechanisms for line and register signalling is different enough that trying to use, for example, Mexican settings for a circuit in Argentina will result in failure. In some countries (such as Brazil) there are even different flavours of the national protocol in use by different carriers.

What this all comes down to is that anyone who wishes to provide hardware or software that connects to an R2 circuit is going to need to understand exactly how the protocol is delivered in the place they are implementing it.

This is why the OpenR2 project is so important.

OpenR2

The OpenR2 project offers a flexible means to ensure that MFC/R2 circuits can be easily connected to open source telephony projects. It also allows for a wide range of configuration options, which ensures that each of the various implementations of the protocol can be readily adapted (including new versions not yet supported).

Currently OpenR2 supports the MFC/R2 standard for the following countries:

Argentina - ar	**China - cn**	**Czech - cz**	**México - mx**	**Phillippines - ph**
Brazil - br	**Colombia - co**	**Ecuador - ec**	**ITU - itu**	**Venezuela – ve**

Note that if you do not see your country listed here, it does not mean that OpenR2 will not work for you. Contact Sangoma for more information.

[1] You will not tend to find R2 circuits in Canada, the USA, or most of Europe, but MFC/R2 is widely used elsewhere in the world, especially in Latin America.

Getting Started with OpenR2

R2 Basics

In an R2 circuit, line signalling is handled by bits on the 16th DS0. Register signalling is accomplished through MF tones. Put more simply, line signalling handles things, like on-hook, off-hook and other details of the state of the channel, while register signalling handles the details of the call, such as who is calling (ANI), and what number they called (DNIS).

MFC/R2 is not a protocol that is likely to become popular in private networks or lab environments, since the same E1 circuit that R2 runs on can be used to run PRI instead (which is technically superior). In the real world, however, R2 circuits are very popular due to cost, and, since they are digital if run over an E1, they offer huge performance improvements over analogue.

R2 and Sangoma

Sangoma is the principal sponsor of the OpenR2 project.

If you have a pair of lab systems and some spare Sangoma T1/E1 cards, it is easy enough to create an R2 circuit to test things out. If you wish to put an R2 circuit into a production environment, it is important to ensure that your national variant is supported.[2]

[2] If in doubt, you should try the ITU variant first.

Installing OpenR2

Installing OpenR2 is not difficult, but because there are a lot of components the task can seem daunting at first. Keep in mind that several of these steps are required simply to install Asterisk® and Zaptel/DAHDI, and you'll realize that OpenR2 hardly adds any effort at all.

We will be performing the following installations:

1. DAHDI-Linux
2. DAHDI-Tools
3. Wanpipe®
4. OpenR2
5. Asterisk®

Only step 4 would not be required in any normal Sangoma/Asterisk install, so you will see very shortly that the OpenR2 installation is very simple to accomplish.

DAHDI-Linux

The Zapata telephony project delivered the first inexpensive telephony interface boards for Open Source telecommunication projects. Quickly adopted by the Asterisk developers, the Zaptel interface became the standard way to connect PSTN circuits to Asterisk. Due to trademark concerns the Zapata library has been re-named DAHDI. Regardless of the name, if you want to connect an Asterisk system to the PSTN using OpenR2, you are going to need to install either DAHDI or Zaptel (depending on which version of Asterisk you are using).

DAHDI needs to be downloaded from the Asterisk website, www.asterisk.org. From the Linux[3] command line you can issue the following commands:

```
# cd /usr/src
# wget http://downloads.digium.com/pub/telephony/dahdi-linux/dahdi-linux-
2.0.0.tar.gz
# tar zxvf dahdi-linux-2.0.0.tar.gz
# cd dahdi-linux-2.0.0
# make
# make install
```

That's all you need to do with DAHDI-Linux for now.

[3] All of the samples in this paper were tested out on CentOS 5.2. The manner in which you install software may be different if you are using another Linux distribution, but in essence, the installation process should be similar.

DAHDI-Tools

Next up are the tools for DAHDI. These used to be part of the Zaptel install, but are now part of a separate package.

```
# cd /usr/src
# wget http://downloads.digium.com/pub/telephony/dahdi-tools/dahdi-tools-
2.0.0.tar.gz
# tar zxvf dahdi-tools-2.0.0.tar.gz
# cd dahdi-tools-2.0.0
# ./configure
# make
# make install
```

The DAHDI tools are now installed.

Wanpipe®

The Wanpipe drivers can seem complex at first, however that is simply due to the fact that this powerful suite of applications is able to serve many other functions other than telephony. For the purposes of OpenR2, the following commands will get the job done:

```
# mkdir /etc/asterisk
# cd /usr/src
# wget ftp://ftp.sangoma.com/linux/current_wanpipe/wanpipe-3.3.14.tgz
# tar zxvf wanpipe-3.3.14.tgz
# cd wanpipe-3.3.14
# ./Setup dahdi
```

Wanpipe will run you through several questions, which you can answer '**Y**'es to. When you arrive at the question "`Would you like to configure wanpipe devices for DAHDI?`" you will again answer '**Y**'es, at which point it will begin the information-gathering process to build the DAHDI config files. In some ways this is not required, since we're going to change these files manually later for R2, but some of the information will remain valid, so you might as well do it now. Accept the defaults unless listed otherwise here.

`Media type = E1`
`Clock = normal`
`Signalling type=FXS loop start` (doesn't really matter since we'll be changing this)
`Full E1=NO` (we'll normally use a fractional E1, but it depends what your carrier is delivering, so this may be `YES`)

We are now ready to install OpenR2.

OpenR2

The OpenR2 project is currently hosted at www.libopenr2.org. Installation of OpenR2 involves the following steps:

```
# cd /usr/src
# wget http://openr2.googlecode.com/files/openr2-1.0.0-rc1.tar.gz
# tar zxvf openr2-1.0.0-rc1.tar.gz
# cd openr2-1.0.0-rc1
# ./configure
# make
# make install
```

The final step in the install process is Asterisk.

Asterisk®

Asterisk requires a special patched version that contains the OpenR2 libraries. As of this writing, the patched asterisk versions are stored on Digium's SVN servers.

Although OpenR2 is developed on Asterisk 1.6, patches are available for both 1.4 and 1.2 as well. The following commands use 1.4.

```
# cd /usr/src
# svn checkout http://svn.digium.com/svn/asterisk/team/moy/mfcr2-1.4
# cd mfcr2-1.4
# ./configure --prefix=/usr
# make
# make install
```

If you want the sample Asterisk files installed, a simple

```
# make config
```

and the installation is done.

Now, on to configuration.

Configuring OpenR2

Now that everything is installed, we have to configure it.

Configuring DAHDI

Because OpenR2 uses configurations that are not used in the stock versions of dahdi, you will need to make changes to both /etc/dahdi/system.conf and /etc/asterisk/chan_dahdi.conf

```
# cd /etc/dahdi
# vim system.conf
```

You will need to edit the file so that it looks something like this:

```
#autogenerated by /usr/sbin/wancfg_dahdi do not hand edit
#autogenrated on 2008-11-24
#Dahdi Channels Configurations
#For detailed Dahdi options, view /etc/dahdi/system.conf.bak
loadzone=us
defaultzone=us

#Sangoma A101 port 1 [slot:0 bus:4 span:1] <wanpipe1>
span=1,0,0,cas,hdb3
cas=1-10:1101
```

Configuring Asterisk

Next, you need to change the parameters under Asterisk for chan_dahdi. Before you do this, you should read the sample file for your region. Examples are located under the folder /usr/src/openr2-1.0.0-rc1/doc/asterisk/

In this example, we will use the data for Mexico, or ‘mx’:

```
# cd /etc/asterisk
# vim chan_dahdi.conf
```

You will want to add the following lines to the end of the file:

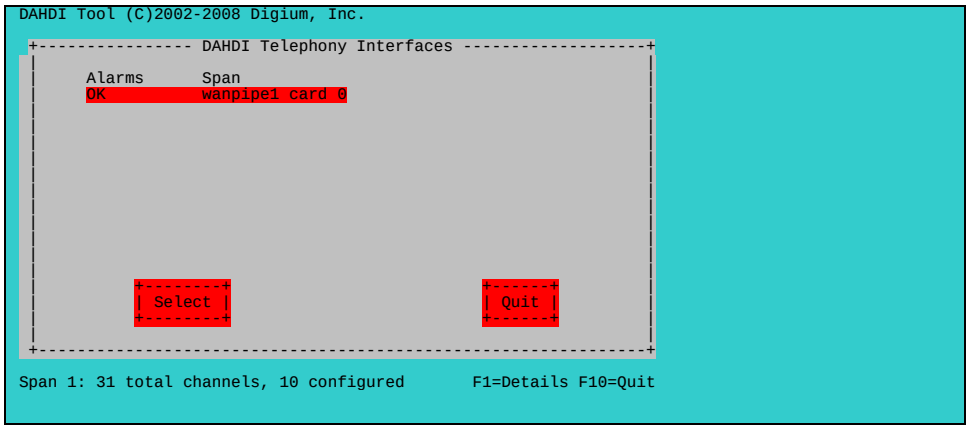
```
signalling=mfcr2
mfcr2_variant=mx
mfcr2_get_ani_first=no
mfcr2_max_ani=10
mfcr2_max_dnis=4
mfcr2_category=national_subscriber
mfcr2_mfback_timeout=-1
mfcr2_metering_pulse_timeout=-1
; this is for debugging purposes
mfcr2_logdir=log
mfcr2_logging=all
; end debugging configuration
channel => 1-10
```

Verify the Installation

Reboot the system and run the following:

```
# dahdi_tool
```

You should see a screen that looks similar to this:



You need to ensure that there are no alarms (make sure your circuit is physically connected), and that the number of configured channels is correct.

Next, you will want to verify that Asterisk has configured the channels correctly and sees them as:

```
# asterisk –rx ‘mfcr2 show channels’
```

You should see something similar to the following:

```
Chan Variant Max ANI Max DNIS ANI First Immediate Accept Tx CAS   Rx CAS
   1 MX      10      4        No        No               IDLE     IDLE
   2 MX      10      4        No        No               IDLE     IDLE
   3 MX      10      4        No        No               IDLE     IDLE
   4 MX      10      4        No        No               IDLE     IDLE
   5 MX      10      4        No        No               IDLE     IDLE
   6 MX      10      4        No        No               IDLE     IDLE
   7 MX      10      4        No        No               IDLE     IDLE
   8 MX      10      4        No        No               IDLE     IDLE
   9 MX      10      4        No        No               IDLE     IDLE
  10 MX      10      4        No        No               IDLE     IDLE
```

For a healthy line, the expected CAS values are "`IDLE`". Any other result, such as "`BLOCK`" or a hexidecimal value means the line is not enabled. Contact your Telco to ask them to enable it.

Your MFC/R2 circuit is now available for use.

Conclusion

There is nothing ‘next generation’ about the MFC/R2 protocols, which have been in use for over 50 years. Nevertheless, the fact that OpenR2 delivers support for this popular technology is yet another example of community-based software delivering practical, inexpensive solutions that are relevant to current needs.

Glossary

Line Signalling

Line signalling handles the state of the circuit at a low level. States, such as, ‘on hook’ and ‘off hook’ are examples of line signalling.

Register Signalling

Register signalling handles the rest of the signalling information; the details of the call. Examples of register signalling are ANI (who is calling) and DNIS (what number was dialed).

DTMF

Dual-Tone Multi Frequency signalling is something anyone with a touch tone phone has heard. This is the signal a telephone generates in order to pass information to a far end (for example an automated attendant that you have dialed).

MFC

Multi Frequency Compelled. This is a mechanism whereby audio tones are sent down telecom circuits in order to convey signalling information (Caller ID, for example).

R2 (and R1)

Strictly speaking, R2 simply refers to Region 2 and R1 is Region 1. Nowadays, protocols have evolved and been renamed, so the term ‘R2’ is most-commonly used as a nickname for the protocol properly known as MFC/R2. The term ‘R1’ is not commonly heard at all anymore, certainly not when describing protocols.

DS0

A DS0 (dee-ess-zero) is a 64-bit timeslot in a digital circuit. It is important because it represents the bandwidth necessary to carry a conversation over a phone line. This is the basic building block of digital telecom circuits, and all telecommunications throughout the world is based on this standard.

MF Tones

Similar to DTMF, Multi Frequency tones are used to pass signalling information between telecommunications switches. This is considered an older style of signalling, as nowadays, this sort of information is passed digitally, however MF remains in use throughout the world.