OWL installation and minimal use guide:

OWL is built on the srsLTE library by Software Radio Systems. Thus, if you are already able to use srsLTE (at least pdsch_ue) OWL should run without many issues. However, this guide will give you the minimum set of information to install all the required software on a clean ubuntu 14.04 installation (it may work on different distribution, but it is not tested). Also, this guide include steps for the preferable installation of the Nuand BladeRF software. OWL is also tested on USRP x310, but the guide for a correct installation of the related UHD library is to be done in the next release.

Part 1 - Installation:

1.1 Install dependencies:
$ sudo apt-get install build-essential git cmake libboost-system-dev libboost-test-dev libboost-thread-dev libqwt-dev libqt4-dev libfftw3-dev

1.2 BladeRF installation (skip this if you use USRP )
https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux

To activate the release PPA, simply:
$ sudo add-apt-repository ppa:bladerf/bladerf
$ sudo apt-get update
$ sudo apt-get install bladerf libbladerf-dev bladerf-firmware-fx3 bladerf-fpga-hostedx40

1.3 Install srsgui (this is not mandatory for OWL to work, but is a nice tool and it helps testing srsLTE and OWL):
$ git clone https://github.com/suttonpd/srsgui.git
$ cd srsgui
$ mkdir build
$ cd build
$ cmake ../
$ make
$ sudo make install

1.4 Install gnuradio (only VOLK is needed, but the VOLK standalone installation usually fails and installing gnuradio in this way is usually fast and error-free. If you have problem with this, just try to have VOLK installed https://github.com/gnuradio/volk):
$ sudo apt-get install gnuradio

1.5 Install OWL
from inside the srsLTE folder do the following:
$ mkdir build
$ cd build
$ cmake ../
$ make

If everything succeeded you will find OWL's executables together with srsLTE's examples in the srsLTE/build/srslte/examples folder.

Part 2 - Executable description:
2.1 Inherited from srsLTE

- cell_search
Scan a given LTE band trying to acquire synchronization with the base station. Please, refer to this website for the used frequencies in your country http://www.spectrummonitoring.com/frequencies

cell_search requires the band number and can also use the EARFCN numbers to narrow the search, please refer to this website for details:
http://niviuk.free.fr/lte_band.php

- pdsch_ue
emulate a UE trying to connecting to a given frequency, first looking for synchronization and, then, decoding control messages related to broadcast transmissions.
In addition, this program provide some useful statistics about synchronization and decoding success rate.
TIP: once you have the frequency of a base station you can run
$ ./pdsch_ue -f <freq>
where <freq> is the base station central frequency in hertz, i.e. 1.8 GHz can be given as 1800e6 or 1.8e9. If the synchronization is successfull, pdsch_ue will plot the constellations of the control channel and the shared downlink channel (only broadcast messages). If the signal is clean, you should be able to see a QPSK constellation in both diagrams. In addition, the amplitude and phase channel responses are plotted together with the PSS synchronization. The last one is ok if it looks like a gaussian.

2.2 OWL files
- imdea_capture_sync
This program capture a raw trace of the LTE channel synchronized on the beginning of the first subframe 0 detected. A very useful reference is
http://www.sharetechnote.com/html/FrameStructure_DL.html
Usage:
$ ./imdea_capture_sync -f <freq> -l <cell_num> -n <subframe_num> -o <output_filename>
<freq> is the base station central frequency in hertz, i.e. 1.8 GHz can be given as 1800e6 or 1.8e9.
<cell_num> is in {0,1,2} and can be obtained from cell_search or pdsch_ue (see for reference http://www.sharetechnote.com/html/Handbook_LTE_PCI.html)
<subframe_num> is the number of subframes to be recorded in the trace. 1 subframe = 1 millisecond
<output_filename> where to record the trace. The trace can be then used with the other programs for offline processing.
TIP: putting -o /dev/null creates no output, but allows to test the signal synchronization without risking any buffer overrun (pc not ready when the sdr sends the streams). The output of the program is one line per frame (10 ms) and can be:
Decoded MIB ... (good)
MIB not decoded ... (noise on the channel)
sync loss (bad)

- imdea_cc_decoder
This program is the main part of OWL, where the control channel is decoded. It works both online and offline and pre-recorded traces.
Online usage:
$ ./imdea_cc_decoder -f <freq> -n <subframe_num> 1> <cc_out_filename> 2> /dev/null
<cc_out_filename> specifies where to save the decoded control channel messages. If omitted, the messages are printed to the stdout. Don't forget to redirect the stderr (2> /dev/null), which is used to produce the list of location to be checked by the fine-tuner.
The output of the decoder is a tab separated list where each line represents a decoded message. The columns are as follows:
1. SFN: internal timing of LTE (1 every frame = 10 ms)
2. subframe index from 0 to 9 (1 subframe = 1 ms)
3. RNTI in decimal
4. Direction: 1 = downlink; 0 = uplink
5. MCS in 0 - 31

6. number of allocated resource blocks in 0 - 110
7. transport block size in bits
8. transport block size in bits (code word 0), -1 if n/a
9. transport block size in bits (code word 1), -1 if n/a
10. DCI message type. This version only scans for 0 (format 0), 2 (format 1a), 6 (format 2a)
11. new data indicator toggle for codeword 0
12. new data indicator toggle for codeword 1
13. HARQ process id
14. ncce location of the DCI message
15. aggregation level of the DCI message
16. CFI
17. DCI correctness check

Offline usage:
$ ./imdea_cc_decoder -i <input_trace_filename> -l <cell_num> -c <pci> -P <ports> -p <prb> -z <rnti_out_filename> -Z <rnti_in_filename> 1> <cc_out_filename> 2> <cc_fix_filename>
<input_trace_filename> a trace that you saved with imdea_capture_sync
<pci> <ports> <prb> physical cell id, antenna ports and number of physical resourrce blocks of the LTE channel. All of these can be obtained using cell_search, pdsch_ue, imdea_capture_sync
<rnti_in_filename> <rnti_out_filename> are the rnti lists. They are optional; if not provided the tool generates a new list. If available it starts with the information given. The format is a vector of 65355 elements, which can be 0 (not used), 1 (used, but not used in the last 10 seconds), 2 (used in the last 10 seconds).
<cc_fix_filename> output file for the fine_tuner program. It specify one location to be checked per line. The columns are as follows:
1. SFN
2. subframe
3. ncce
4. L
5. CFI

- imdea_fine_tuner
offline tool to post_process recorded trace to try to decode DCI messages in location that could not be decoded by imdea_cc_decoder.
Usage:
$ ./imdea_fine_tuner -i <input_trace_filename> -l <cell_num> -c <pci> -P <ports> -p <prb> -z <cc_fix_filename> -Z <rnti_in_filename> 1> <cc_fixed_filename> 2> /dev/null
it can only be used after imdea_cc_decoder on the output produced. imdea_fine_tuner generate <cc_fixed_filename> with the same format of <cc_out_filename> (see above)

Part 3 - OWL setup and first use

0. Install everything correctly!
1. Use a software defined radio capable of sampling at a LTE compliant sampling rate (30.72, 23.04, 15.36, 11.52).
2. srsLTE and OWL are heavy on the I/O. Try not to have read/write operations at the same time (different buffers may interfer). If overruns are detected, consider working on ramdisk (see https://wiki.archlinux.org/index.php/Tmpfs)
3. Use a good antenna, capable of LTE frequencies. Check the bands used in your country on http://www.spectrummonitoring.com/frequencies
4. Use cell_search to get the central frequencies of the available cell in the surrounding
5. Try to get a map of the base station location and put your sniffer in a location where a line of sight with the antenna is available

6. Use pdsch_ue and imdea_capture_sync to assess the signal quality
7. Once you have a good signal, you should have pdsch_ue showing very clean QPSK constellations and imdea_capture_sync showing almost only "Decoded MIB ..." output.
8. At this point you should have <freq> <cell_num> <pci> <ports> <prb>
9. Try first the online decoder with the output on screen:
$ ./imdea_cc_decoder -f <freq> -n <subframe_num> 2> /dev/null
10. Try a capture, with subsequent decoding and fine tuning:
$ ./imdea_capture_sync -f <freq> -l <cell_num> -n <subframe_num> -o <output_filename>
$ ./imdea_cc_decoder -i <input_trace_filename> -l <cell_num> -c <pci> -P <ports> -p <prb> -z <rnti_out_filename> -Z <rnti_in_filename> 1> <cc_out_filename> 2> <cc_fix_filename>
$ ./imdea_fine_tuner -i <input_trace_filename> -l <cell_num> -c <pci> -P <ports> -p <prb> -z <cc_fix_filename> -Z <rnti_in_filename> 1> <cc_fixed_filename> 2> /dev/null
$ sort -u <cc_out_filename> <cc_fixed_filename> -o <cc_total_filename> (to combine the output)
11. You can also use the recorded trace to obtain a spectrogram of the LTE transmission (in a future release, I will provide matlab and octave scripts to do that as well).
12. Have fun!