LMS8001 Python Package
pyLMS8001

Introduction

Python package pyLMS8001 is platform-independent, and is intended for fast prototyping and algorithm development. It provides low level register access and high level convenience functions for controlling the LMS8001 chip and evaluation boards. Supported evaluation boards are:

- LMS8001_EVB
- LMS8001 Companion Bard

The package consists of Python classes which correspond to physical or logical entities. For example, each module of LMS8001 is a class. The LMS8001 chip is also a class containing instances of on-chip modules. The evaluation board class contains instances of on-board chips, such as LMS8001, ADF4002, etc. Classes follow the hierarchy and logical organization from evaluation board down to on-chip register level.

Installation

The py8001 package is installed in a usual way:

```
python setup.py install
```

Module installation can be verified from Python:

```
python
```

```
>>> from pyLMS8001 import *
```

If there is no error, the module is correctly installed.

Basic usage

The first step is to connect to the evaluation board:

```
>>> from pyLMS8001 import *
```

List of COM ports with LMS8001 boards attached can be obtained as:

```
>>> boards = LMS8001_EVB.findLMS8001()
```

```
>>> lms8001_evb = LMS8001_EVB(boards[0])
```

Now that the board is connected, the on-board chips can be used. For example, board clock can be synchronized to external 10 MHz reference by configuring the on-board ADF4002.

```
>>> adf4002 = limeSDR.ADF4002
```

```
>>> adf4002.enable() # Configure and enable the on-board ADF4002
```

The ADF4002 can be disabled with:

```
>>> adf4002.disable() # Disable the on-board ADF4002
```

On-board LMS8001 chip can be accessed as:

```
>>> lms8001 = lms8001_evb.LMS8001
```

Registers can be accessed with overloaded [ ] operator:

```
>>> lms8001[0x0f]
Register : ChipInfo 0x000F
VER<4:0>            01000               (0x0008 << 11) (8 << 11)
REV<4:0>                 00001          (0x0001 << 6)  (1 << 6)
MASK<5:0>                     000000    (0x0000 << 0)  (0 << 0)
Register value      0100000001000000    (0x4040)
```

Registers can be accessed by address as shown in the previous example, or by name:

```
>>> lms8001['ChipInfo']
```

Register definition can be accessed with the help function:

```
>>> lms8001['ChipInfo'].help()
REGISTER    ChipInfo    0x000F
    BITFIELD   VER<4:0>
        POSITION=<15:11>
        DEFAULT=01000
        MODE=RI
        #!  Chip version.
        #!      01000 - Chip version is 8.
    ENDBITFIELD
    BITFIELD   REV<4:0>
        POSITION=<10:6>
        DEFAULT=00001
        MODE=RI
        #!  Chip revision.
        #!      00001 - Chip revision is 1.
    ENDBITFIELD
    BITFIELD   MASK<5:0>
        POSITION=<5:0>
        DEFAULT=000000
        MODE=RI
        #!  Chip mask.
        #!      000000 - Chip mask is 0.
    ENDBITFIELD
ENDREGISTER
```

Individual bit-fields can be accessed also:

```
>>> chipInfo=lms8001['ChipInfo']
>>> chipInfo['REV<4:0>']
    1
```

Register value can be written directly:

```
>>> lms8001['GPIOOutData']=0xAA
```

Single bitfield can also be changed:

```
>>> lms8001['GPIOOUT_SEL0']['GPIO4_SEL<2:0>']=0
```

Each module in LMS8001 has an instance for each channel. For example, LMS8001 channel A can be accessed:

```
>>> CHA = lms8001.CHANNEL['A']
```

Configuration 0 of channel A mixer B can be set as:

```
>>> CHA.PD[0].MIXB_LOBUFF_PD=0     # Turn on the mixer LO buffer
```

Configuration 0 of channel A power amplifier can be set as:

```
>>> CHA.PD[0].PA_PD=0              # Turn on the PA
```

Channel configuration can be printed by:

```
>>> lms8001.infoChannel('A')
----------------------------------------
|      Channel A PD configuration      |
----------------------------------------
| N | LNA |  MIXA |  MIXB | R50 |  PA  |
----------------------------------------
| 0 | OFF |  OFF  |   ON  | OFF |  ON  | <- Active
| 1 | OFF |  OFF  |  OFF  | OFF |  OFF |
| 2 | OFF |  OFF  |  OFF  | OFF |  OFF |
| 3 | OFF |  OFF  |  OFF  | OFF |  OFF |
----------------------------------------

----------------------------------------
|     Channel A LNA configurations     |
----------------------------------------
| N | GAIN | CGS | ICT_MAIN | ICT_LIN |
----------------------------------------
| 0 |   8  |  2  |    16    |    16   | <- Active
| 1 |   8  |  2  |    16    |    16   |
| 2 |   8  |  2  |    16    |    16   |
| 3 |   8  |  2  |    16    |    16   |
----------------------------------------
```

```
--------------------------------------
|     Channel A PA configurations     |
--------------------------------------
| N | MAIN | LIN | ICT_MAIN | ICT_LIN |
--------------------------------------
| 0 |   0  |  0  |    16    |    16   | <- Active
| 1 |   0  |  0  |    16    |    16   |
| 2 |   0  |  0  |    16    |    16   |
| 3 |   0  |  0  |    16    |    16   |
--------------------------------------

ICC(LNA)  = 0.0  mA
ICC(MIXA) = 0.0  mA
ICC(MIXB) = 25.0 mA
ICC(PA)   = 20.0 mA
Channel A power 26.026 mW
```

Channel info shows that active configuration has mixer B and PA turned on, with LNA and mixer A turned off. Active configuration of power down can be set to configuration #3 with:

```
>>> CHA.PD_INT_SEL=3
>>> lms8001.infoChannel('A')
---------------------------------------
|      Channel A PD configuration      |
---------------------------------------
| N | LNA |  MIXA |  MIXB | R50 |  PA  |
---------------------------------------
| 0 | OFF |  OFF  |   ON  | OFF |  ON  |
| 1 | OFF |  OFF  |  OFF  | OFF |  OFF |
| 2 | OFF |  OFF  |  OFF  | OFF |  OFF |
| 3 | OFF |  OFF  |  OFF  | OFF |  OFF | <- Active
---------------------------------------

--------------------------------------
|     Channel A LNA configurations    |
--------------------------------------
| N | GAIN | CGS | ICT_MAIN | ICT_LIN |
--------------------------------------
| 0 |   8  |  2  |    16    |    16   | <- Active
| 1 |   8  |  2  |    16    |    16   |
| 2 |   8  |  2  |    16    |    16   |
| 3 |   8  |  2  |    16    |    16   |
--------------------------------------
```

```
--------------------------------------
|     Channel A PA configurations     |
--------------------------------------
| N | MAIN | LIN | ICT_MAIN | ICT_LIN |
--------------------------------------
| 0 |   0  |  0  |    16    |    16   | <- Active
| 1 |   0  |  0  |    16    |    16   |
| 2 |   0  |  0  |    16    |    16   |
| 3 |   0  |  0  |    16    |    16   |
--------------------------------------

ICC(LNA)  = 0.0  mA
ICC(MIXA) = 0.0  mA
ICC(MIXB) = 0.0  mA
ICC(PA)   = 0.0  mA
Channel A power 0.0 mW
```

High level functions

Besides the basic functionality for reading/writing registers, high level functions are also provided to simplify the chip configuration. For example, configuring and locking the PLL to a given frequency requires a sequence of steps. The pyLMS8001 package provides the high level functions for the following operations:

- Configuration and locking of PLL
- Chip configuration from ini files generated by LMS8001 GUI

PLL can be configured and locked to a given frequency, in this case 6 GHz, with a single command:

```
>>> lms8001.PLL.frequency=6.0e9
```

LO distribution should be enabled for channel to operate:

```
>>> lms8001.PLL.setLODIST(channel="A", EN=1, IQ=False, phase=0)
```

```
>>> lms8001.PLL.setLODIST(channel="C", EN=1, IQ=False, phase=180)
```

Chip configuration can be read from ini file and programmed into LMS8001 with:

```
>>> lms8001.readIniFile('chipConf.ini')
```