# Overview of RegControl Package for Python

The RegControl Python API consists of a set of Python and C++ based functions for controlling registers using tokens and formulas. In particular, the C++ based functions are placed in a DLL and the Python ctypes package is used to access these DLL functions from Python.

The key C++ DLL files are:

* reg\_control.dll
  + Low level C++ routines to read and write register settings
    - Author: Michael Perrott
    - This is provided as an example file with empty functions GetReg() and SetReg(). It is assumed that the user would fill in appropriate code for the GetReg() and SetReg() functions.
* token\_code\_lib.dll
  + Low level C++ routines to perform token related operations
    - Author: Michael Perrott
    - Source code framework: two C++ source files:
      * token\_code.cpp (and token\_code.h)
        + Core routines for token based control of registers
      * token\_code\_lib.cpp
        + Wrapper routines to allow interfacing to Python

The key Python files are:

* reg\_control.py
  + Python wrapper functions to reg\_control.dll which also utilize token\_code\_lib.dll and token\_code.py
    - Author: Michael Perrott
    - Source code framework: one Python file that contains RegControl Python class from which all methods are accessed.
* token\_code.py
  + Python wrapper functions to token\_code\_lib.dll as well as supporting routines to perform token mapping and formula calculations.
    - Author: Michael Perrott
    - Source code framework: one Python file that contains TokenMap Python class from which all methods are accessed.

Supporting Python files are:

* reg\_control\_gui\_functions.py
  + Helper GUI function of a Multi-Column Listbox that is useful for displaying register values.
    - Author: Michael Perrott

There are also some test scripts that provide useful examples of using the Python methods:

* test\_reg\_control.py
  + Example of using the RegControl class.
    - Author: Michael Perrott
* test\_token\_code.py
  + Example of using the TokenMap class
    - Author: Michael Perrott

# RegControl

This section provides further details on reg\_control.py

## RegControl Class

### Class definition, initialization, closure:

* Class definition: RegControl()
  + Creates RegControl object from which all methods in this section are called
  + Example: rtc = RegControl()

### Class methods:

* Register read and write operations
  + getreg(reg\_name)
    - Gets token or inv\_formula result for input register field name
    - Inputs:
      * register field name (string)
    - Output:
      * token value of register field (string)
  + setreg(reg\_name, token\_val)
    - Sets token or formula input for input register field name
    - Inputs:
      * Register field name (string)
      * Token value or formula input (string)
    - Output: none
  + getreg\_raw(reg\_name)
    - Gets decimal value for input register field name
    - Inputs:
      * Register field name (string)
    - Output:
      * Decimal value of register field (integer)
  + setreg\_raw(reg\_name, data\_val)
    - Sets decimal value for input register field name
    - Inputs:
      * Register field name (string)
      * Decimal value of register field (integer)
    - Output: none
* Getting information on registers, tokens, formulas
  + get\_reg\_names(reg\_type='register')
    - Obtain a list of all register field names or test block names
    - Inputs:
      * Register type, which can be ‘register’ or ‘test\_block’ (string)
    - Output:
      * List of strings corresponding to register names
  + get\_token\_names(reg\_name, reg\_type='register')
    - Obtain a list of token names for a given register field or test block
    - Inputs:
      * Register type, which can be ‘register’ or ‘test\_block’ (string)
    - Output:
      * List of strings corresponding to token names
  + get\_pad\_info(test\_block\_name, token\_name)
    - Obtain pad information corresponding to a given token name of a given test block
    - Inputs:
      * Test block name (string)
      * Token name of the given test block (string)
    - Output:
      * 4-character string corresponding to pad configuration (string)
  + get\_matching\_token(reg\_name, input\_value)
    - Get token matching an input token for a given register field. Note that this method is mainly useful for error checking
    - Inputs:
      * Register field name (string)
      * Input token value (string)
    - Output:
      * In case of success, output token is same as input token (string)
      * In case of failure, output is ‘error\_token\_not\_found’ (string)
  + get\_nearest\_token(reg\_name, input\_value)
    - Get token that is the closest numerical match to the tokens associated with a given register field. It is assumed that all token are numerical, though single character scale suffixes are OK (example, ‘10k’, ‘20m’,’30M’,’50p’)
    - Inputs:
      * Register field name (string)
      * Input numerical token value (string)
    - Outputs:
      * Token value that is numerical closest to input (string)
  + get\_reg\_info(reg\_name, info\_str)
    - Get information on given register field, with such information being the options ‘default\_str’, ‘description’, ‘num\_bits’, and ‘signed\_val’
    - Inputs:
      * Register field name (string)
      * Requested information (string)
        + ‘default\_str’: default register decimal value (in the case of formula being used)
        + ‘description’: description of register
        + ‘num\_bits’: number of bits in register field
        + ‘signed\_val’: 1 if signed formula, 0 otherwise
    - Output:
      * Information as noted above (string)
  + max\_formula\_input(reg\_name)
    - Convenience function for determining formula input corresponding to maximum register field value. Note that maximum value for register field value depends on whether formula is signed or unsigned and the number of bits in the register field
    - Inputs:
      * Register field name (string)
    - Output:
      * Formula input to achieve maximum register field value (string)
  + min\_formula\_input(reg\_name)
    - Convenience function for determining formula input corresponding to minimum register field value. Note that minimum value for register field value depends on whether formula is signed or unsigned and the number of bits in the register field
    - Inputs:
      * Register field name (string)
    - Output:
      * Formula input to achieve minimum register field value (string)
* Print operations for register, token, and pad information and general debugging
  + print\_reg(reg\_field\_search\_name)
    - Print register field names that match the input substring in a case insenstitive manner
    - Inputs:
      * Register field substring, with examples being:
        + rtc.print\_reg(‘’): print all register field names
        + rtc.print\_reg(‘del’): print all register field names containing substring ‘del’ (matched in a case insensitive manner)
    - Output: none (just print out statements)
  + print\_reg\_tokens(reg\_field\_search\_name)
    - Print tokens associated with a given register field name that match the input substring in a case insensitive manner
    - Inputs:
      * Register field substring (similar to print\_reg() above)
    - Output: none (just print out statements)
  + print\_reg\_state(reg\_field\_search\_name)
    - Print current state of register fields that match the input substring in a case insensitive manner
    - Inputs:
      * Register field substring (similar to print\_reg() above)
    - Output: none (just print out statements)
  + print\_reg\_state\_gui(reg\_field\_search\_name)
    - GUI version of print\_reg\_state()
  + print\_test(test\_block\_search\_name)
    - Print test block names that match the input substring in a case insensitive manner
    - Inputs:
      * Test block substring, with examples being:
        + rtc.print\_test(‘’): print all test block names
        + rtc.print\_test(‘bias’): print all test block names containing substring ‘bias’ (matched in a case insensitive manner)
    - Output: none (just print out statements)
  + print\_test\_tokens(test\_block\_search\_name)
    - Print tokens associated with a given test block name that match the input substring in a case insensitive manner
    - Inputs:
      * Test block substring (similar to print\_test() above)
    - Output: none (just print out statements)