SPIBoy User Guide

Installation

Virtual Environment

It's recommended distribute a python app using **virtual environment** for better portability and consistency. A virtual environment is a container isolated from the global Python environment, which means it contains and only contains a selected set of packages necessary for the program to run.

This repository is already constructed using virtual environment. To enable the venv, execute the following command in CLI:

```
$ source bin/activate
```

Manual Installation

Required Package:

- [x] **tabulate**: Used for ASCII table generation

```
$ pip install tabulate
```

Additional Packages:

- [x] **ipython**: Interactive CLI
- [x] **jupyter**: Enhanced iPython
- [ ] **cython**: Python-to-C Compiler

```
$ pip install ipython jupyter
```

Data Types

Right now, all the input data is recorded by python **dictionary** data type written directly in python source file. I'm considering JSON (which is quite popular for ascii structured data exchange I recon) for future versions.

---

spiParam

A dictionary of all the interface parameters. Mostly used by the ASCII output functions (write CSV, etc)

Key	Type	Used ?	Description
wordsize	int	UNUSED	how many bits are in each SPI transaction

Key	Type	Used ?	Description
shortidsize	int	UNUSED	how many bits are in the short ID
longidsize	int	UNUSED	how many bits are in the long ID
cmdsize	int	UNUSED	how many bits are in the SPI command word
payloadsize	int	UNUSED	how many bits are in the SPI payload word
cpol	int	UNUSED	Clock Polarity, 1 or 0
cpha	int	UNUSED	Clock Phase, 1 or 0
cidle	int	USED	Clock Idle Logic Level, 1 or 0
ce_active_high	bool	USED	CE is active high
rst_active_high	bool	USED	RESET is active high
autoce	bool	UNUSED	toggle CE automatically in the *SPI.w* function call
LSBfirst	bool	USED	If false, swap bit sequence when at output
clk_pin	str	USED	The name of the clock pin, used for ascii output
rst_pin	str	USED	The name of the reset pin, used for ascii output
ce_pin	str	USED	The name of the ce pin, used for ascii output
other_pins	list of str	USED	A list of names of all auxiliary pins available for the *SPI.pinDeposit* function call

spiCmdDict

A dictionary of all the SPI commands

Entry Format: `Command Name (str) : Command Word (int)`

Example:

```
cmdDict = {
    'LOAD_POINTER' : 0b0000000,
    'READ_REGISTER': 0b0000001,
    ...
}
```

spiRegDict

A dictionary of all the SPI Addresses

Entry Format: `[ l1 (list), l2 (list), Info (str), Omit Config (int)]`

L1 ACACIA Register Entry: `[Addr (int), RW (str), Base (int), Word Size (int)]`

- Addr: ACACIA Address
- RW: Designating the R/W property of the register, 'rw', 'w' or 'r'
- Base & Word Size: The actual valid portion of the register, example:
    - Base = 10, Word Size = 10 means the valid portion of the register is wire[9:0] = reg[19:10]

L2 SPI Register Entry: `[Addr (int), RW (str), Base (int), Word Size (int)]`

Same as above, the primary difference is that a SPI register is 16-bit wide, but an ACACIA register is 128-bit wide

If, say for example, a register is accessible via SPI but not ACACIA. The list L1 is set to []

Info: A string describing the register. Useful for documentation

Omit Config: telling the interpreter which part of the register information can be omitted. For example, to generate a more concise human-readable output, we want to turn the following output:

```
 SPI      | Name   | RW     | ACACIA   | ACACIA | SPI     | Text
 Addr     |        |        | Addr     | Mask   | Mask    |
----------+--------+--------+----------+--------+---------+------
 0x0000   | reg1   | rw     | 0x0020   | [12:0] | [12:0]  | Text1
 0x0001   | reg2   | rw     | 0x0021   | [12:0] | [12:0]  | Text2
 0x0002   | reg3   | rw     | 0x0022   | [12:0] | [12:0]  | Text3
 0x0003   | reg4   | rw     | 0x0023   | [12:0] | [12:0]  | Text4
 0x0004   | reg5   | rw     | 0x0024   | [12:0] | [12:0]  | Text5
 0x0005   | reg6   | rw     | 0x0025   | [12:0] | [12:0]  | Text6
```

Into this:

```
 SPI      | Name   | RW     | ACACIA   | ACACIA | SPI     | Text
 Addr     |        |        | Addr     | Mask   | Mask    |
----------+--------+--------+----------+--------+---------+------
 0x0000   | reg1   | rw     | 0x0020   | [12:0] | [12:0]  | Text1
           ... (2 registers omitted) ...
 0x0003   | reg4   | rw     | 0x0023   | [12:0] | [12:0]  | Text4
 0x0004   | reg5   | rw     | 0x0024   | [12:0] | [12:0]  |
 0x0005   | reg6   | rw     | 0x0025   | [12:0] | [12:0]  |
```

The omit config associated with each register should be:

Register	Omit Config	Function
reg1,3	0	No Omit
reg2	1	Omit All
reg4,5	2	Omit Help Text

Methods of the SPI2CSV class

SPI = SPI2CSV.SPI2CSV(spiParam, cmdDict, regDict)

Instantiating function *__init__*: load the three dictionaries

SPI2CSV.i2hexStr(i:int) -> str

Convert an integer to a 32bit hex string (in '0x??' form)

If spiParam['LSBfirst'] == True, the bit sequence will be inverted

SPI2CSV.writeRegDict(fname:str, dict:dict, target:str, omitEnable:bool)

Output the dictionary in one of the following formats:

- **target**='readable': Human readable
- **target**='csv': CSV file, intened for documentation

SPI2CSV.w(id, cmd, data, mask, clkc, comment, IDType])

Add a entry in the internal buffer that performs an SPI transaction

- **id**: chip ID
- **cmd**: command str
- **data**: payload str
- **mask**: Some part of the final DIN data field can be marked as "modifiable", which means that GF can change it on the fly. For example, for a 32-bit transcation, a mask of 0xFFFF0000 means that DIN[31:16] are marked as editible
- **clkc**: How many clocks to run for this transaction. The default value is 32
- **comment**: A comment string
- **IDType**: Not implemented yet. When IDType=0 the transaction should be 64-bit, in which 32 bits are the long chip ID

SPI2CSV.pinDeposit(pinDict)

Add a entry in the internal buffer that changes the internal state of pins, does not write directly to the final CSV file

- **pinDict**: A {str:bool} dictionary of all the pins that will be changed

SPI2CSV.wCommentLine(line)

Add a entry in the internal buffer that writes a comment line. (i.e. the entire line is occupied by a single comment, all the other fields are left empty)

- **line**: A comment string

SPI2CSV.wReset(clkc, comment)

Add a entry in the internal buffer that issues a reset. The reset will last *clkc* clock cycles

SPI2CSV.writeCSV(fname)

Commit the internal buffer to the final CSV file requested by GF