PCIe400: Drivers I2C

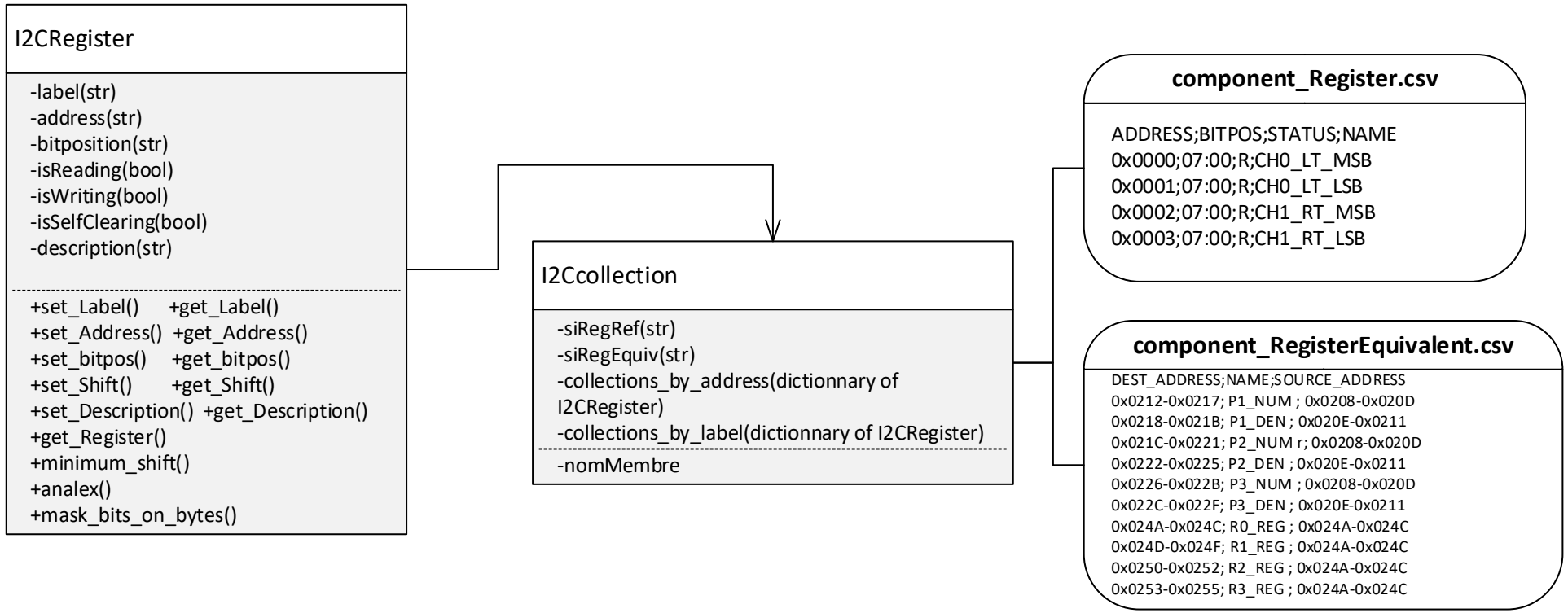
pour les composants dédiés de la carte

- Architecture
- Principe
- Développement en cours

Architecture

- Les composants I2C sélectionnés:
  - De quelques dizaines de registres
  - Jusqu'à plusieurs centaines de registres
  - Plusieurs façons d'utiliser le composant suivant les paramètres des registres
- Difficultés:
  - Les cas d'utilisations
    - Besoin d'entrées des testeurs de la carte
  - Généralisation des actions quelques soit le composant
- Solution:
  - Chaque driver garde en mémoire l'état du composant
  - Chaque changement vers le matériel est sauvegardé
  - Architecture adaptable à tous composants utilisant le ftdi
  - Architecture évolutif (nvlles fonctions) en fonction des besoins

Architecture



Classe **Registre**

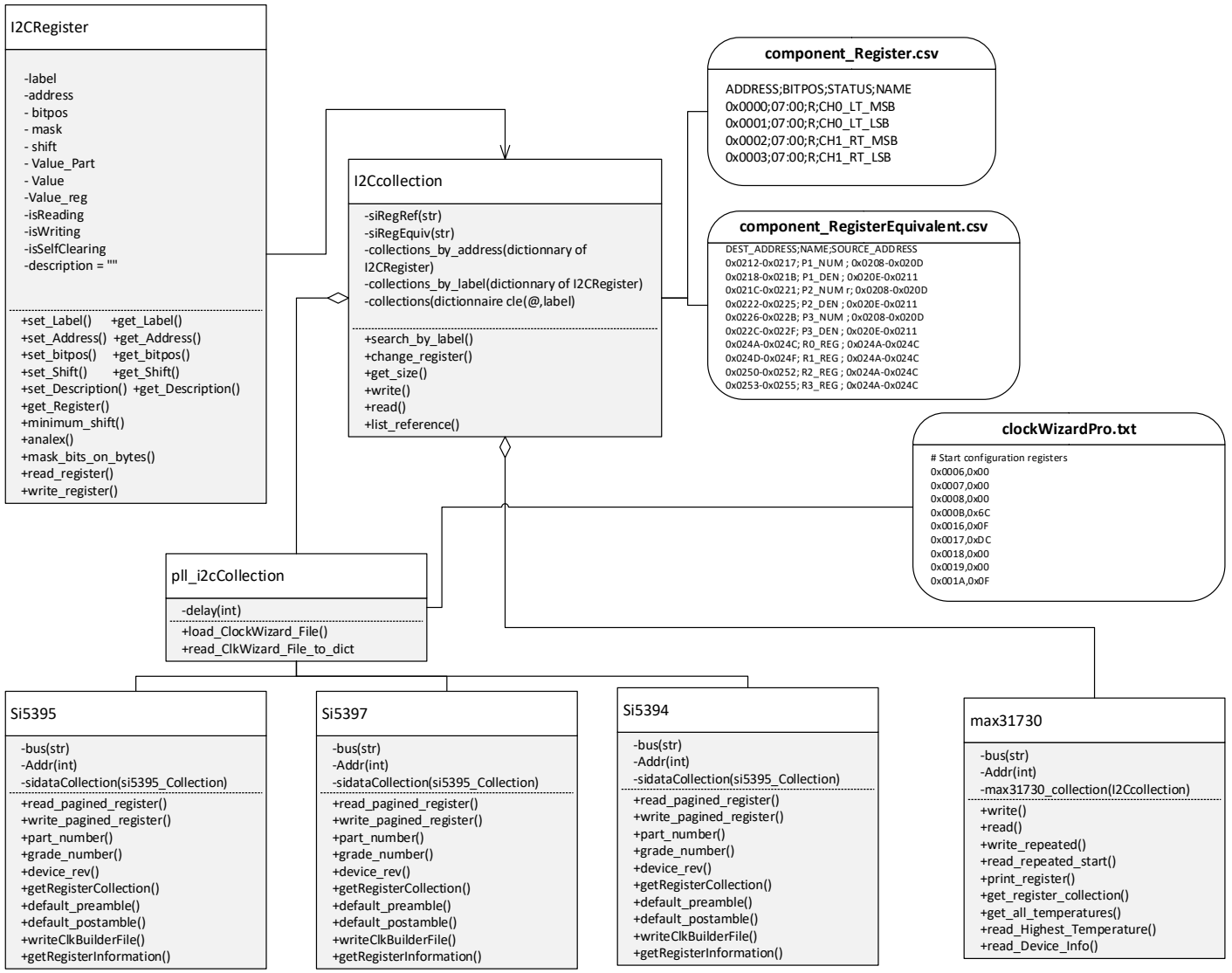
Ensemble des actions que l'on peut définir sur un registre

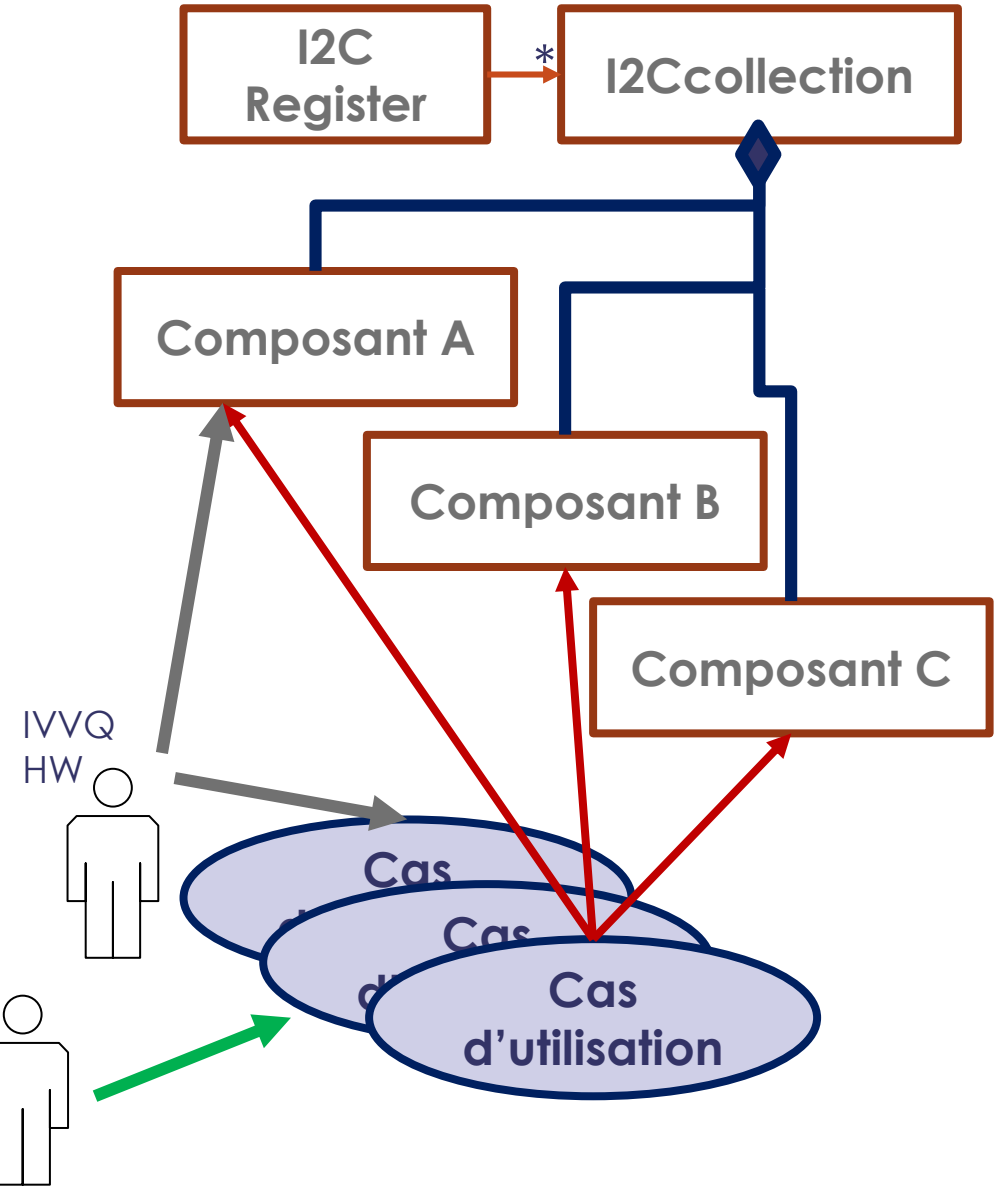
Classe **collection: ensemble de registres**

Sauvegarde de l'état d'un composant

Fichier csv (txt)
déclaration des registres d'un composant

Principe: Puissance de l'héritage





Fonctions spécialisées + Fonctions génériques

Exemples

```python
from si5394 import SI5394

bus = "ftdi://ftdi:232h:FT61Z4FQ/1"
add = 0x68
component_si5394 = SI5394(bus, add)
myreg = component_si5394.search_by_address("0x002E")
dict_test = myreg[0].get_register()


file = "../components/test_clkWpro.txt"
component_si5394.load_clockwizard_file(file)
myreg = component_si5394.search_by_address("0x002E")

several_reg = component_si5394.read_pagined_register("PN_BASE_LSB")
for reg_read in several_reg:
    val = reg_read.get_value_reg()
    print(several_reg.get_label() + ": " + str(val))

several_reg = component_si5394.write_pagined_register(
        "OOF1_RATIO_REF_LSB", 0x10
    )
several_reg =
component_si5394.read_pagined_register("OOF1_RATIO_REF_LSB")
or reg_read in several_reg:
    val = reg_read.get_value_reg()
    print(several_reg.get_label() + ": " + str(val))


result = component_si5394.part_number()
print(result + " : " + "5394")
```

```
# dict_test = {
#         "Label": "LOS0_TRG_THR_LSB",
#         "Address": 46,
#         "page": 0,
#         "status": {"R": True, "W": True, "S": False},
#         "mask": 255,
#         "Shift": 0,
#         "partValue": 0,
#         "partialValue": 0,
#         "completeValue": 0,
#     }
```

Exemples

```
import I2Ccollection as i2cc
import pytest
import logging


temp_f1 = "../components/test_Register.csv"
temp_f2 = "../components/test_equiv.csv"
temp_file = [temp_f1, temp_f2]
regCollection_100 = i2cc.I2Ccollection(temp_file)


dict_resp = regCollection_100.search_by_label("PAGE")
page_reg = dict_resp[0]
page_reg.set_value(0x0000)
page_reg.write_register(monbus, monaddr)
testReg = regCollection_100.read(monbus, monaddr, "PN_BASE")
logging.debug(testReg)
i = 0
test = ["0x94", "0x53"]
for reg in testReg:
    val = regCollection_100.get_collection()[reg].get_value_reg()
    assert hex(val) == test[i]
    i = i + 1
```

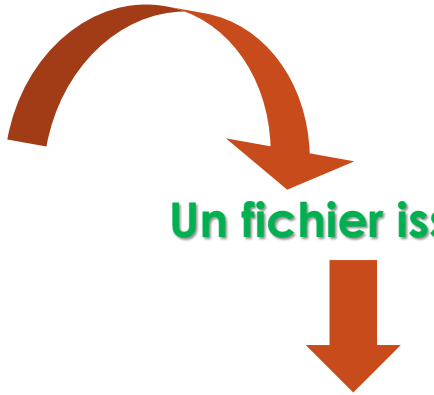
PMBUS

❑ Spécifications:

The three components are based on ***PMBUS*** standard. It means each register address could respond to 8 actions:
1-send Command
2-write bytes
3-write word
4-group
5-read bytes
6-read word
7-read block
8-write block
Each command could control *one* bytes, *two* bytes or *N* bytes as parameters.
The final user could access each register, write/read if available. But, it will access macro functions like VOUT reading, STATUS reading, Temperature reading.
The three components has the same king of registers to control except some marginal difference in number. Foreseen class:
PMBUSREGISTER
PMBUSCOLLECTION
LTM4681
LTM4677
LTC2975

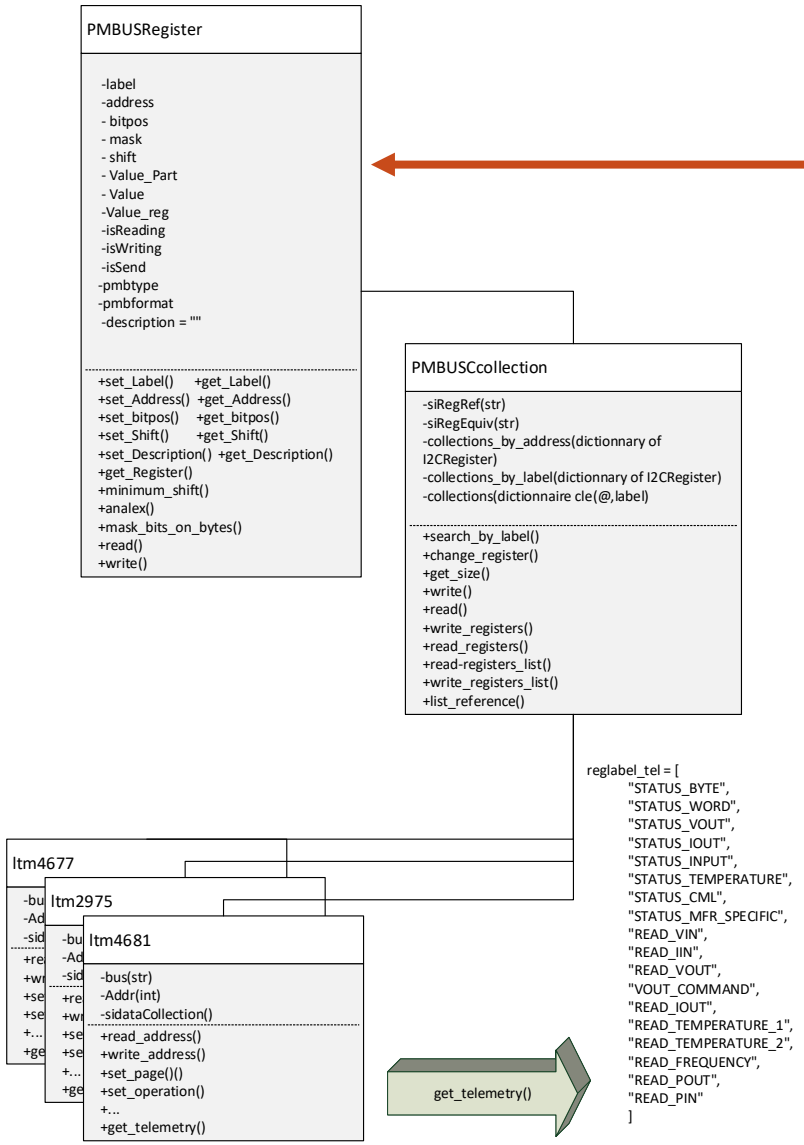
PMBUS Architecture

The class is a register, storing information on one PMBUS register.
a register has the following properties:
- **a label**
- an address
- a mask
- its complete value from reading or writing. **Value** is an iterable like a List.
- a flag to check its status in reading, writing, self-clearing, sending, block of value
-The **TYPE** of information which is stored in the register:
  -Byte : 8-bit reg
  -Word : 16-bit reg
  -Block: A serie of 8-bit values
  -string: 8bit ascii code
-The **format** of the stored data:
  -Reg: 8bit information
  -L11 : fixed point information
  -L16 : Fixed point information
  -ASC : ASCII format
-a description of its action if needed



Un fichier issus de la « datasheet »

COMMAND NAME	CMD CODE	STATUS	TYPE	PAGED	DATA FORMAT	UNITS	NVM	DEFAULT VALUE
PAGE	0x0000	R/W	Byte	N	Reg	0	0	0x00
OPERATION	0x0001	R/W	Byte	Y	Reg	0	Y	0x80
ON_OFF_CONFIG	0x0002	R/W	Byte	Y	Reg	0	Y	0x1E
CLEAR_FAULTS	0x0003	Send	Byte	N	0	0	0	NA
PAGE_PLUS_WRITE	0x0005	W	Block	N	0	0	0	
PAGE_PLUS_READ	0x0006	R/W	Block	N	0	0	0	
WRITE_PROTECT	0x0010	R/W	Byte	N	Reg	0	Y	0x00
STORE_USER_ALL	0x0015	Send	Byte	N	0	0	0	
RESTORE_USER_ALL	0x0016	Send	Byte	N	0	0	0	
CAPABILITY	0x0019	R	Byte	N	Reg	0	0x0000B0	
SMBALERT_MASK	0x001B	R/W	Block	Y	Reg	0	Y	CMD
VOUT_MODE	0x0020	R	Byte	Y	Reg	0	2â€“12 0x0014	
VOUT_COMMAND	0x0021	R/W	Word	Y	L16	V	Y	1.0 x1000
VOUT_MAX	0x0024	R/W	Word	Y	L16	V	Y	3.6 0xC399
VOUT_MARGIN_HIGH	0x0025	R/W	Word	Y	L16	V	Y	1.05
VOUT_MARGIN_LOW	0x0026	R/W	Word	Y	L16	V	Y	0.95
VOUT_TRANSITION_RATE	0x0027	R/W	Word	Y	L11	V/ms	Y	0.25
FREQUENCY_SWITCH	0x0033	R/W	Word	N	L11	kHz	Y	350kHz
VIN_ON	0x0035	R/W	Word	N	L11	V	Y	4.75
VIN_OFF	0x0036	R/W	Word	N	L11	V	Y	4.5




```
status = {"R": True, "W": True, "S": False}
test_reg = PMBUSRegister("PAGE", 0x0003,
status, "Byte", "Reg")
print(
    test_reg.get_label(),
    " :",
    test_reg.get_address(),
    " :",
    test_reg.get_value(),
    " : ",
    test_reg.get_rawaddress(),
    " : ",
    test_reg.get_bitpos(),
)

print(test_reg.get_register(False))
```

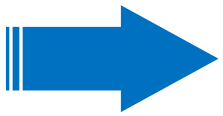
Exemples

```
    op = ltm4681user.read(self.bus, self.addr, label="CAPABILITY",
                          readlen=1, start=start, relax=relax)
```

```
reglabel_tel = [
            "STATUS_BYTE",
            "STATUS_WORD",
            "STATUS_VOUT",
            "STATUS_IOUT",
            "STATUS_INPUT",
            "STATUS_TEMPERATURE",
            "STATUS_CML",
            "STATUS_MFR_SPECIFIC",
            "READ_VIN",
            "READ_IIN",
            "READ_VOUT",
            "VOUT_COMMAND",
            "READ_IOUT",
            "READ_TEMPERATURE_1",
            "READ_TEMPERATURE_2",
            "READ_FREQUENCY",
            "READ_POUT",
            "READ_PIN"
            ]
```

```
local_telemetry = []
local_telemetry = ltm4681user.read_registers_list(self.bus, self.addr,
                                                  reglabel=reglabel_tel,
                                                  start=True, relax=False)
```

```
local_telemetry = []
local_telemetry = ltm4681user.read_registers(self.bus, self.addr,
                                             label=« Vout »,
                                             start=True, relax=False)
```



LTpowerPlay® GUI

DC1613 USB-to-PMBus converter