PCIe400: Drivers I2C

pour les composants dédiés de la carte

- Architecture
- Principe
- Développement en cours

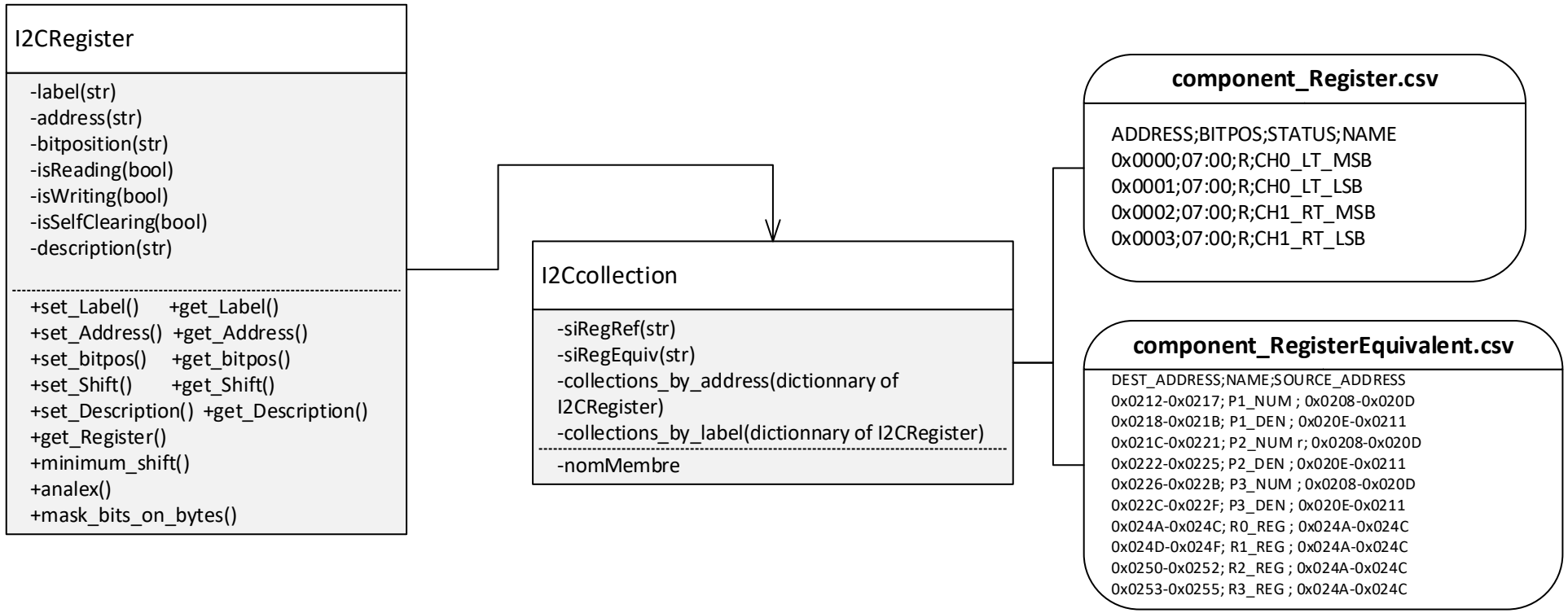
Architecture

- 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



Classe **Registre**

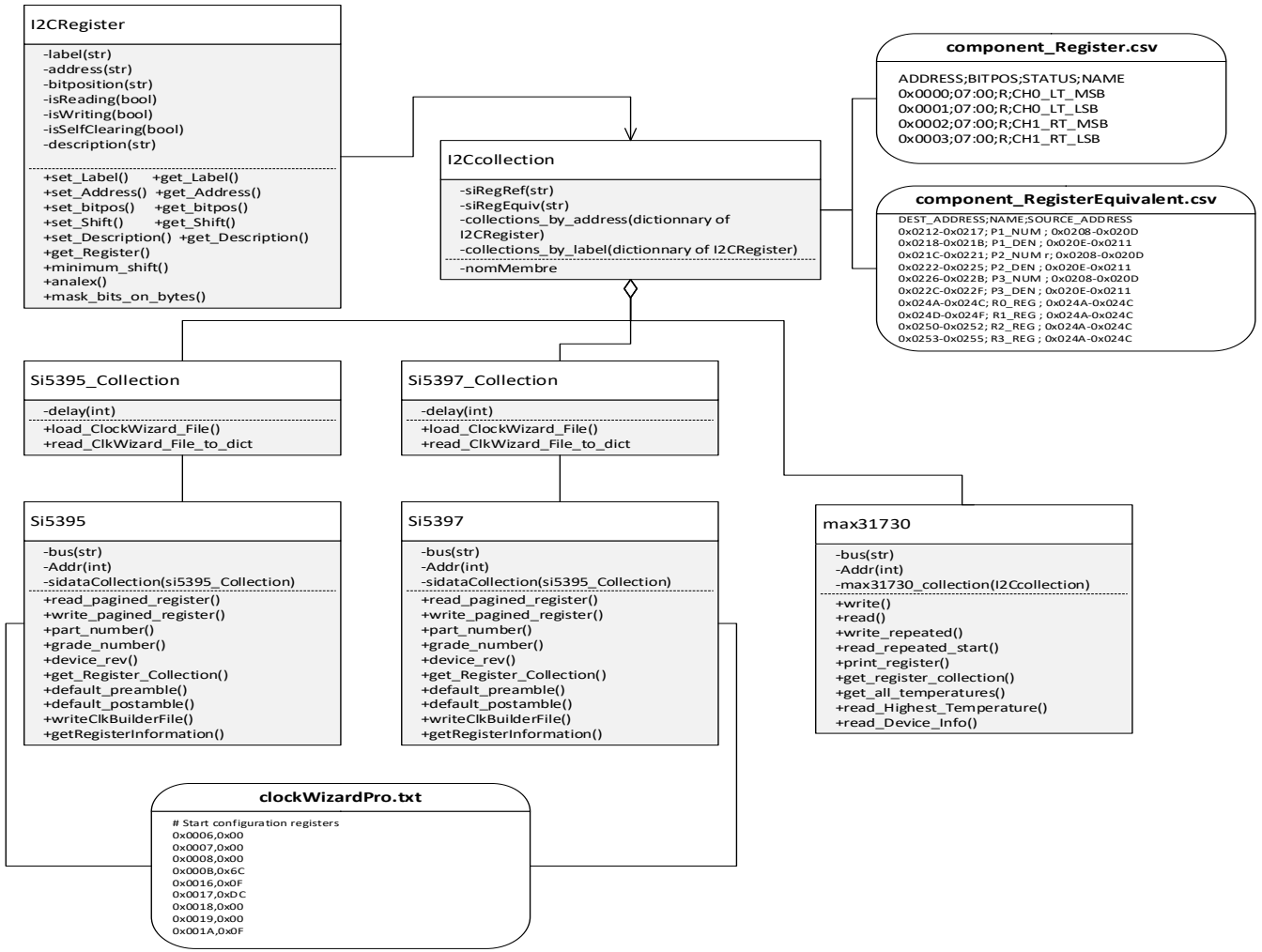
Ensemble des actions que l'on peut définir sur un registre

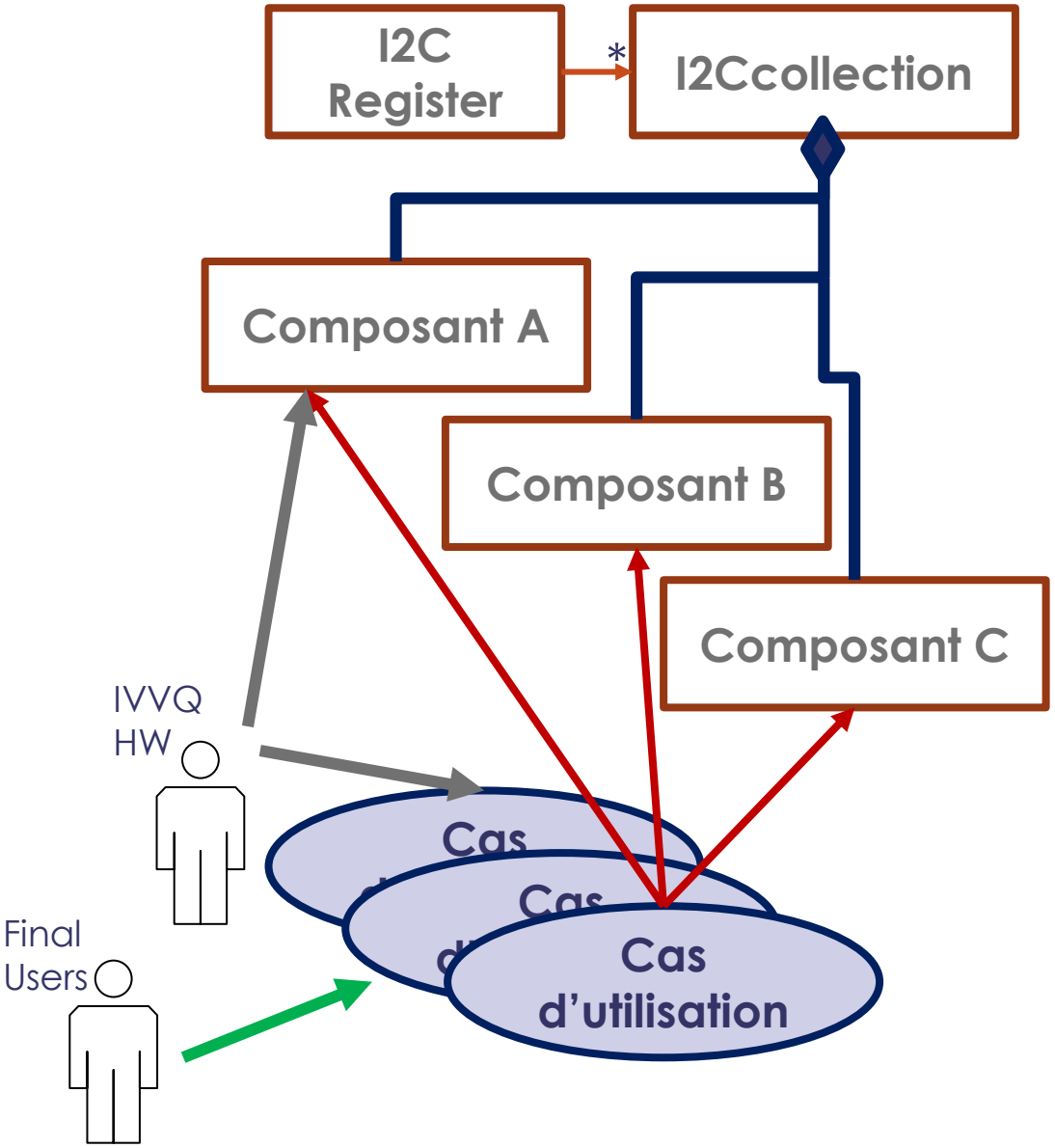
Classe **collection: ensemble de registres**

Sauvegarde de l'état d'un composant

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

Principe: Puissance de l'héritage





Architecture : classe I2CRegister

Reg Address	Bit Field	Type	Name	Description
0x0041	4:0	R/W	OOF0_DIV_SEL	Sets a divider for the OOF circuitry for each input clock 0,1,2,3. The divider value is $2^{OOFx\_DIV\_SEL}$. CBPro sets these dividers.
0x0042	4:0	R/W	OOF1_DIV_SEL	
0x0043	4:0	R/W	OOF2_DIV_SEL	
0x0044	4:0	R/W	OOF3_DIV_SEL	
0x0045	4:0	R/W	OOFXO_DIV_SEL	

plusieurs registres I2C pour une information partielle

Reg Address	Bit Field	Type	Name	Description
0x003E	7:4	R/W	LOS_MIN_PERIOD_EN	Set by CBPro

1 registre I2C pour une information partielle

Reg Address	Bit Field	Type	Name	Description
0x003C	7:0	R/W	LOS3_CLR_THR	16-bit Threshold Value
0x003D	15:8	R/W	LOS3_CLR_THR	

2 registres I2C pour la même fonction

Un registre I2C est sur 8 bits
Un nom de registre différent
Une position des bits modulo 8

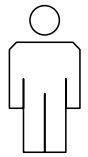
LOS3_CLR_THR_LSB
LOS3_CLR_THR_MSB

Bitpos = 15:8 → 7:0

Valeurs dans un registre:
Valeur particulière<n:0>
Valeur partielle<m:l>
Valeur de registre<7:0>

Exemple: SI5395

IVVQ HW



Description de la fonction à implementer

```
def active_input(self):
    """Read current active input.

def set_active_input(self):
    """Set active input.

def status(self, level="minor"):
    """Get status flags of PLL in a dictionary.

def info(self, level="minor"):
    """Print status of PLL in a dictionary.

def info_frequency_plan(self):
    """Print frequency plan.

def output_setting(self, channel):
    """Read output setting for a single output channel.

def phase_shift(self, ndivider, command):
    """Order N-divider to shift it phase.

def _nx_divider(self, channel):
    """Read N-divider parameters.

def _r_divider(self, channel):
    """Read R-divider.

def _fvco(self):
    """Read Fvco.
```

```
def status(self, level="minor"):
    """Get status flags of PLL in a dictionary.

    Flags are boolean status information active high.

    Note:
        Make use of I2C block mode by reading several registers in a row to
        minimize number of access.

    - SMBUS_TIMEOUT: SMBus timeout error
    - LOSXAXB: Loss of signal reference crystal
    - SYSINCAL: System in calibration
    - OOF_IN0: Out of frequency input 0
    - OOF_IN1: Out of frequency input 1
    - OOF_IN2: Out of frequency input 2
    - OOF_IN3: Out of frequency input 3
    - LOS_IN0: Loss of signal input 0
    - LOS_IN1: Loss of signal input 1
    - LOS_IN2: Loss of signal input 2
    - LOS_IN3: Loss of signal input 3
    - HOLD: DSPLL in holdover or free run mode
    - LOL: Loss of lock
    - CAL_PLL: DSPLL in calibration
    - SMBUS_TIMEOUT_STICKY: Sticky version of SMBus timeout error
    - LOSXAXB_STICKY: Sticky version of loss of signal reference crystal
    - SYSINCAL_STICKY: Sticky version of system in calibration
    - OOF_IN0_STICKY: Sticky version of out of frequency input 0
    - OOF_IN1_STICKY: Sticky version of out of frequency input 1
    - OOF_IN2_STICKY: Sticky version of out of frequency input 2
    - OOF_IN3_STICKY: Sticky version of out of frequency input 3
    - LOS_IN0_STICKY: Sticky version of loss of signal input 0
    - LOS_IN1_STICKY: Sticky version of loss of signal input 1
    - LOS_IN2_STICKY: Sticky version of loss of signal input 2
    - LOS_IN3_STICKY: Sticky version of loss of signal input 3
    - HOLD_STICKY: Sticky version of DSPLL in holdover or free run mode
    - LOL_STICKY: Sticky version of loss of lock
    - CAL_PLL_STICKY: Sticky version of DSPLL in calibration

    Args:
        level {"minor", "normal", "extra"}: level of information to collect

    Returns:
        dict{str;int}: dictionary of flags read

    Raises:
        P400ModelException

    """
```

Exemple: SI5395 ➔ Utilisation

classe **SI5395.py** → *monComposant = SI5395() # instance de la classe SI5395*

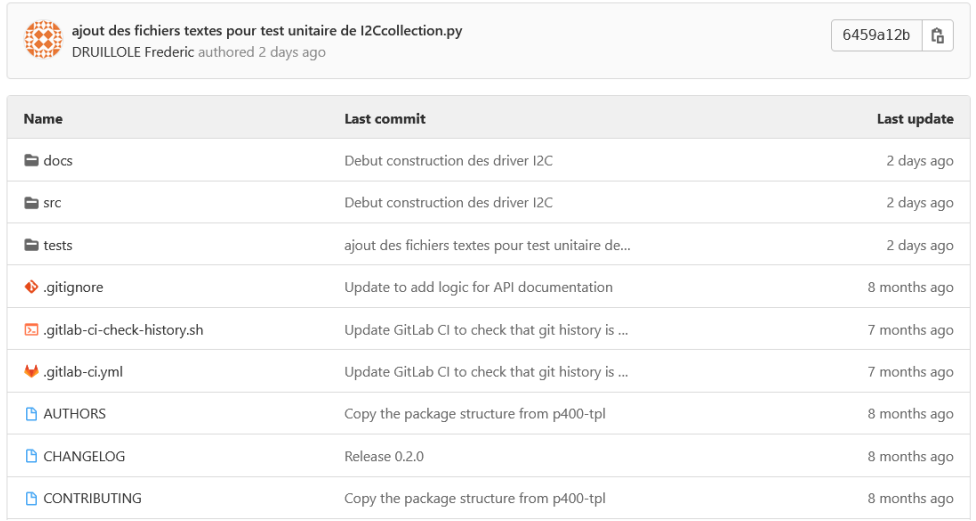
- *Création de la collection « SI5395 Registers »*
- *Chaque action réalisée est sauvegardée dans la collection*
- *Représente l'état du composant au niveau matériel*
- *Sauvegarde de la collection pour réutilisation (sérialisation)*

Développement en cours

SI5395 : PLL
SI5397: PLL
MAX31730 : 4 voies Température

- Classe par composant
- Test Unitaires par classe

- Test avec matériel
  - OK pour le SI5395



➔Définir une stratégie de test avec le matériel
➔Definir une stratégie de repertoire dans GitLab
  ➔1 repertoire par composant ?
  ➔1 seul repertoire pour l'ensemble des composants ?