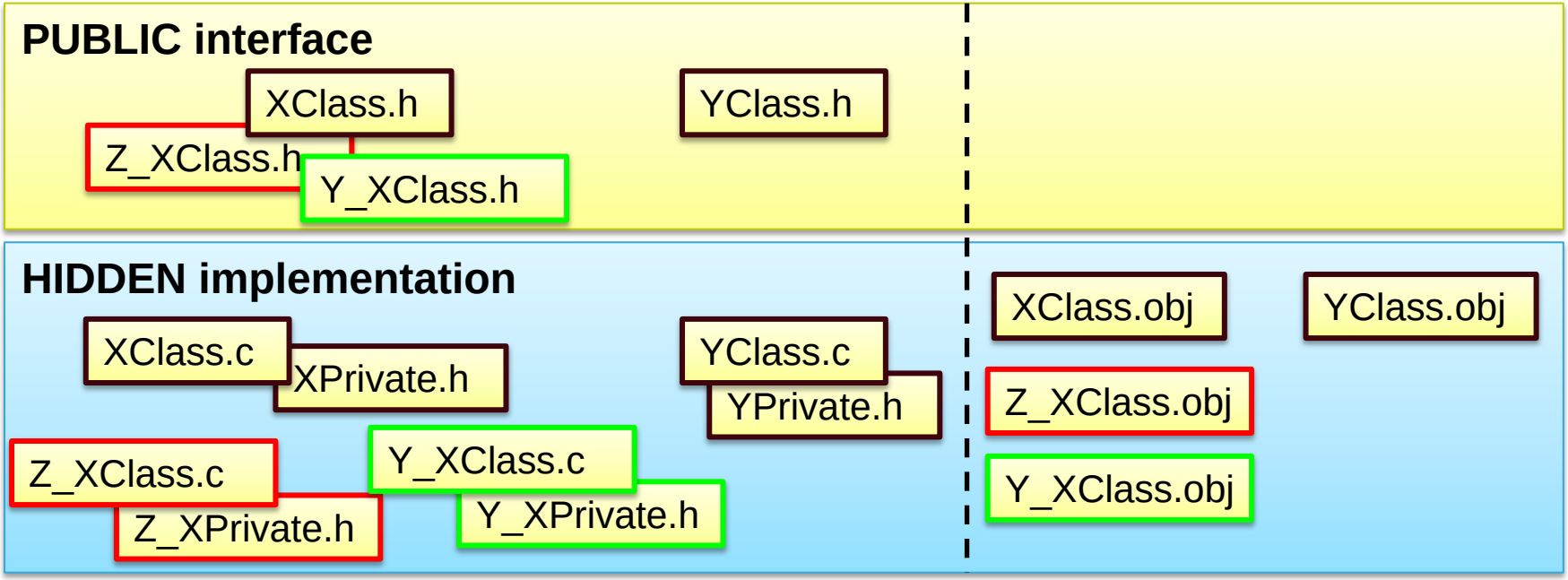
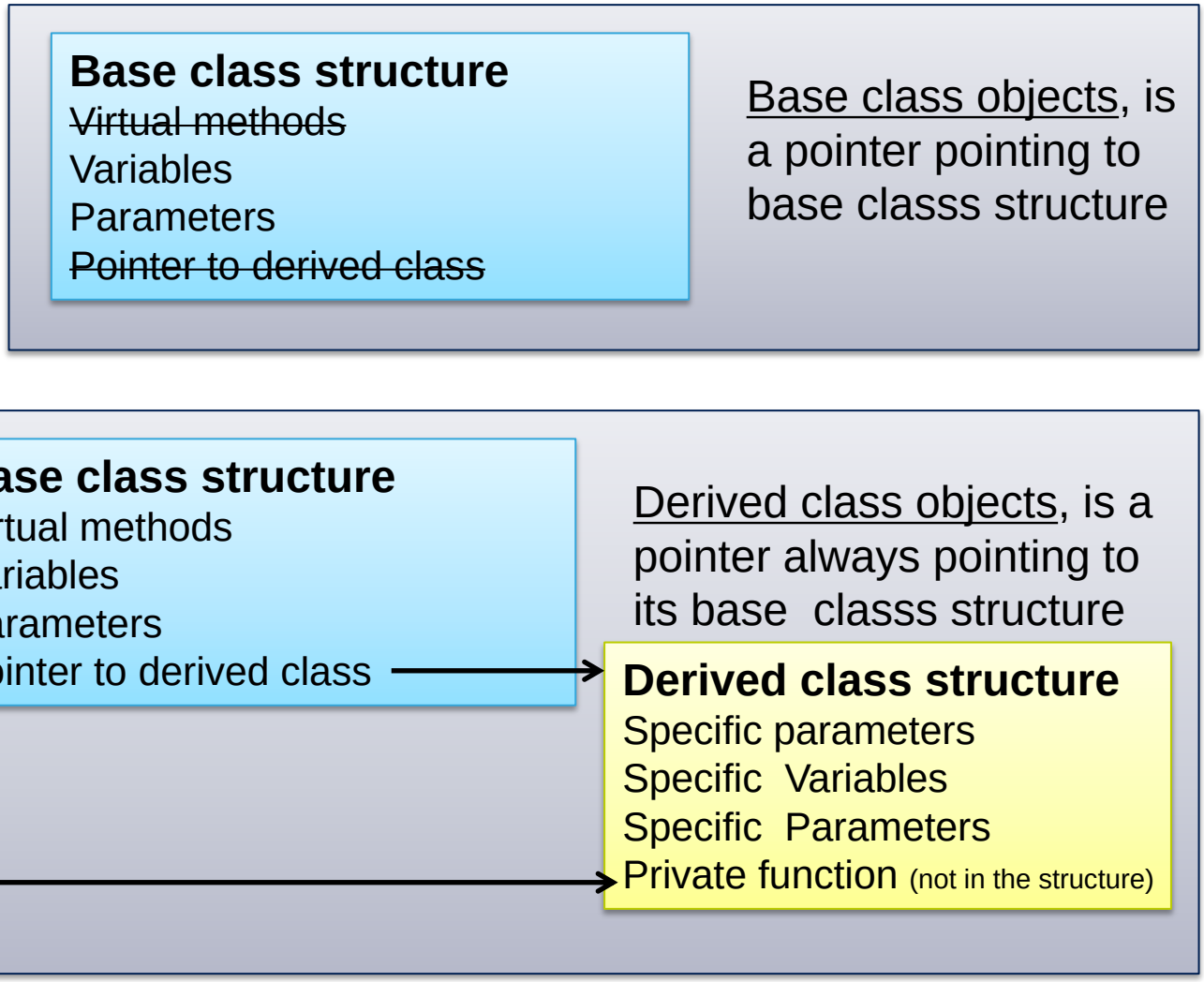
Our implementation of OOP

- It's a C implementation, not really object programming
- Each class is constituted by three source file:
  - An interface header file containing only what it necessary to the user to work with that class
  - A private header file containing private definitions (e.g. object variables, object data structure type)
  - A private .c file containing the real implementation of the methods



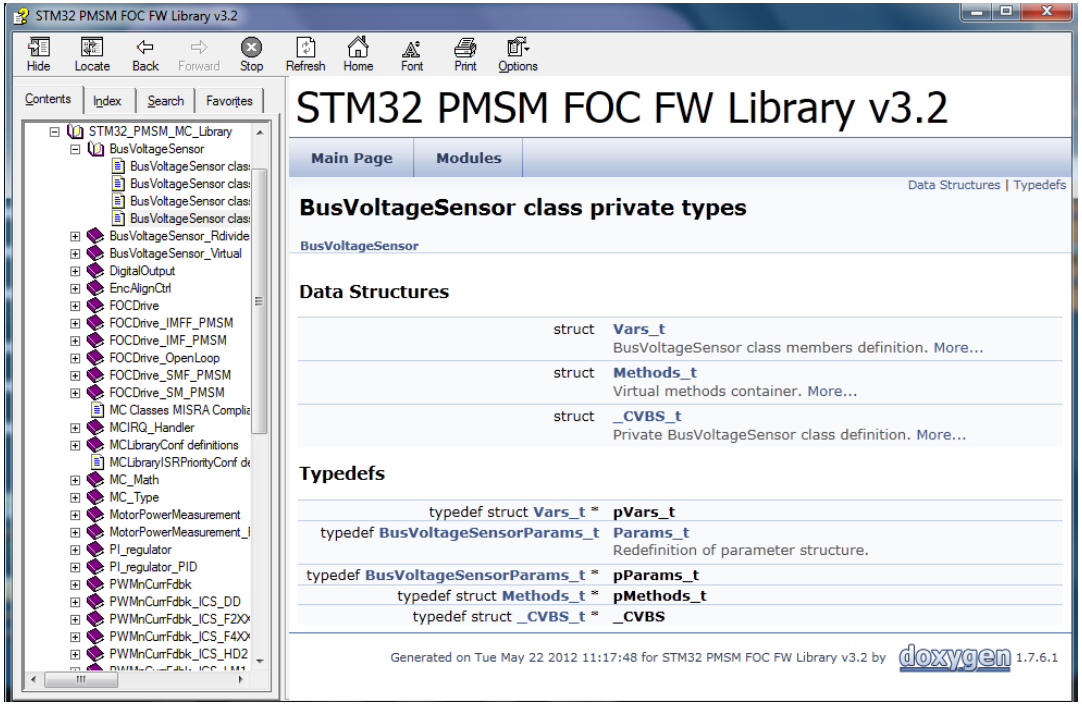
Our implementation of objects relations



Classes interfaces documentation

- As the classes interface is the only file may be known by the user, it has been documented using doxygen format

- .chm help files are then generated for each class for user convenience (as done for the Std peripheral library)



Base classes interface – BCLASSNAMEClass.h

- The interface header file containing only what it necessary to the user to work with that class:

```
typedef struct CPWMC_t *CPWMC;
```

Public class definition, just a void container to allow data hiding

```
typedef const struct
{
  uint16_t hPWMperiod;     /*!< It contains the PWM period expressed
                                in timer clock cycles unit:
                               hPWMPeriod = Timer Fclk / Fpwm    */
}PWMnCurrFdbkParams_t, *pPWMnCurrFdbkParams_t;
```

Parameters structure type, a structure containing the set of all those constant values necessary and sufficient to characterize an object of that class

```
void PWMC_Init(CPWMC this, void *pDrive);
uint16_t PWMC_CheckOverCurrent(CPWMC this);
void PWMC_ADC_SetSamplingTime(CPWMC this, ADConv_t
ADConv_struct);
```

Class methods, the operations on the object internal state are only allowed through these functions

Base classes private definitions – BCLASSNAMEPrivate.h

- The private header file include BCLASSNAMEClass.h and contains private type definitions:

```
#define SECTOR_1      0u
```

Constants definitions only exported to derived classes

```
typedef struct
{
  uint16_t  hT_Sqrt3;   /*!< Contains a constant utilized by SVPWM algorithm */
  uint16_t  hSector;    /*!< Contains the space vector sector number */
              …
}Vars_t,*pVars_t;
```

Object variables structure definition

```
typedef PWMnCurrFdbkParams_t Params_t, *pParams_t;
```

Private redefinition of parameters structure

```
typedef struct
{
  void *(*pIRQ_Handler)(void *this, unsigned char flag);
                                  …
uint16_t (*pPWMC_IsOverCurrentOccurred)(CPWMC this);
} Methods_t,*pMethods_t;
```

Virtual methods container (the first being an interrupt handler if required by object implementation)

```
typedef struct
{
  Methods_t Methods_str;      /*!< Virtual methods container */
  Vars_t Vars_str;                              /*!< Class members container */
  pParams_t pParams_str;      /*!< Class parameters container */
  void *DerivedClass;                           /*!< Pointer to derived class */
}_CPWMC_t, *_CPWMC;
```

Actual class definition, includes pointer to derived class

Base classes private implementation – BCLASSNAME.c

- The private base class implementation file include both BCLASSNAMEClass.h and BCLASSNAMEPrivate.h (can then access both variables and parameters) and actually implements methods

```
#ifdef MC_CLASS_DYNAMIC
#include "stdlib.h" /* Used for dynamic allocation */
#else
```

Inclusion for dynamic memory allocation (objected located in heap and created on demand, not MISRA compliant)

```
#include "MC_type.h“
#define MAX_PWMC_NUM 1u
#else
_CPWMC_t PWMCpool[MAX_PWMC_NUM];
unsigned char PWMC_Allocated = 0u;
#endif
```

Inclusion for static memory allocation(fixed number of objects located in static RAM)

```
#define SQRT3FACTOR (uint16_t) 0xDDB4
```

Private constant definition

```
void PWMC_GetPhaseCurrents(CPWMC this,Curr_Components* pStator_Currents)
{
  (((_CPWMC)this)->Methods_str.pPWMC_GetPhaseCurrents)(this, pStator_Currents);
}
```

Virtual methods implementation (call derived class method)

```
uint16_t PWMC_SetPhaseVoltage(CPWMC this, Volt_Components Valfa_beta)
  {
            ...
 }
```

Real methods implementation

Derived classes interface – DCLASSNAME_BCLASSNAMEClass.h

- The interface header file containing only what it necessary to the user to work with that class. It includes

```
typedef struct CR1LM1_PWMC_t *CR1LM1_PWMC;
```

Public class definition, just a void container

```
typedef const struct
{
  uint32_t wADC_Clock_Divider;    /*!< APB2 clock prescaling factor for
                                   ADC peripheral. It must be RCC_PCLK2_DivX
                                   x = 2, 4, 6, 8 */
  uint8_t IRQnb;                  /*!< MC IRQ number used for TIM1 Update event
                                   and for DMA TC event.*/
                  …
  uint16_t hBKINPin;
} R1_LM1Params_t, *pR1_LM1Params_t;
```

Derived class parameters type, a structure containing a set of constant values that together with base class parameters are necessary and sufficient to characterize an object of this class

```
CR1LM1_PWMC R1LM1_NewObject(pPWMnCurrFdbkParams_t
pPWMnCurrFdbkParams, pR1_LM1Params_t pR1_LM1Params)

static void R1LM1_StartTimers(void)
```

Specific derived class methods

Derived classes private definitions – DCLASSNAME_BCLASSNAMEPrivate.h

- The private header file include BCLASSNAMEClass.h and contains private type definitions:

```
typedef struct
{
  uint16_t hPhaseOffset;      /*!< Offset of current sensing network  */
  uint16_t hDmaBuff[2];
                  …
uint16_t hRegConv;
} DVars_t,*pDVars_t;
```

Specific derived class variables structure definition

```
typedef R1_LM1Params_t DParams_t, *pDParams_t;
```

Private redefinition of derived class parameter structure

```
typedef struct
{
                DVars_t DVars_str;
                pDParams_t pDParams_str;
}_DCR1LM1_PWMC_t, *_DCR1LM1_PWMC;
```

Private derived class definition (no pointer to further derived classes)

Derived classes private implementation – DCLASS_NAME_BCLASSNAME.c

- The private derived class implementation file include both base and derived class interface and private header file (can then access to both derived and base classes variables and parameters) and actually implements real and virtual methods

```
#define ADC1_DR_Address     0x4001244Cu
```

Private constants definitions

```
#ifdef MC_CLASS_DYNAMIC
    #include "stdlib.h" /* Used for dynamic allocation */
#else
    _DCR1LM1_PWMC_t R1LM1_PWMCpool[MAX_R1LM1_PWMC_NUM];
   unsigned char R1LM1_PWMC_Allocated = 0u;
#endif
```

Inclusion for dynamic and static memory allocation

```
static uint16_t R1LM1_CalcDutyCycles(CPWMC this);
static void R1LM1_GetPhaseCurrents(CPWMC this,Curr_Components* pStator_Currents);
```

Private declaration of functions implementing virtual methods

```
CR1LM1_PWMC R1LM1_NewObject(pPWMnCurrFdbkParams_t
pPWMnCurrFdbkParams, pR1_LM1Params_t pR1_LM1Params)
{
                    …
}
```

Derived class specific methods mplementation

```
static void R1LM1_GetPhaseCurrents(CPWMC this,Curr_Components*
pStator_Currents)
 {
            ...
}
```

Function implementing base class virtual methods

New Object implementation – base class


```
#define MAX_PWMC_NUM 1u
#ifdef MC_CLASS_DYNAMIC
#include "stdlib.h" /* Used for dynamic allocation */
#else
_CPWMC_t PWMCpool[MAX_PWMC_NUM];
unsigned char PWMC_Allocated = 0u;
#endif


CPWMC PWMC_NewObject(pPWMnCurrFdbkParams_t pPWMnCurrFdbkParams)
{
  _CPWMC _oPWMC;

#ifdef MC_CLASS_DYNAMIC
  _oPWMC = (_CPWMC)calloc(1u,sizeof(_CPWMC_t));
#else
  if (PWMC_Allocated  < MAX_PWMC_NUM)
  {
    _oPWMC = &PWMCpool[PWMC_Allocated++];
  }
  else
  {
    _oPWMC = MC_NULL;
  }
#endif

  _oPWMC->pParams_str = (pParams_t)pPWMnCurrFdbkParams;
  return ((CPWMC)_oPWMC);
}
```

New Object implementation – derived class


```
#ifdef MC_CLASS_DYNAMIC
   #include "stdlib.h" /* Used for dynamic allocation */
#else
   _DCR1LM1_PWMC_t R1LM1_PWMCpool[MAX_R1LM1_PWMC_NUM];
  unsigned char R1LM1_PWMC_Allocated = 0u;
#endif
CR1LM1_PWMC R1LM1_NewObject(pPWMnCurrFdbkParams_t pPWMnCurrFdbkParams, pR1_LM1Params_t
                                                    pR1_LM1Params)
{
  _CPWMC _oPWMnCurrFdbk;
  _DCR1LM1_PWMC _oR1_LM1;
  _oPWMnCurrFdbk = (_CPWMC)PWMC_NewObject(pPWMnCurrFdbkParams);
 #ifdef MC_CLASS_DYNAMIC
   _oR1_LM1 = (_DCR1LM1_PWMC)calloc(1u,sizeof(_DCR1LM1_PWMC_t));
 #else
    if (R1LM1_PWMC_Allocated  < MAX_R1LM1_PWMC_NUM)
    _oR1_LM1 = &R1LM1_PWMCpool[R1LM1_PWMC_Allocated++];
    else          _oR1_LM1 = MC_NULL;
#endif

_oR1_LM1->pDParams_str = pR1_LM1Params;
_oPWMnCurrFdbk->DerivedClass = (void*)_oR1_LM1;

_oPWMnCurrFdbk->Methods_str.pIRQ_Handler = &R1LM1_IRQHandler;
Set_IRQ_Handler(pR1_LM1Params->IRQnb, (_CMCIRQ)_oPWMnCurrFdbk);
 _oPWMnCurrFdbk->Methods_str.pPWMC_Init = &R1LM1_Init;
...
```

Base class new object creation

Derived class new object creation

Derived and base classes objects linking

Virtual methods linking

Interrupt handling 1/2

- Some derived classes (e.g. R1_LM1 class, ENC, …) need to execute some code on Interrupt Service Routines (ISR)
- The ISR itself can't be moved into objects implementation to allow user adding his own code on the same interrupt
- A special class (**MCIRQHandler**) is created to this purpose
- This class contains an MC interrupt data table which is filled (through Set_IRQ_Handler method) with a set of objects
- As already seen, the pointer to the base class object structure coincides with the address of the MC virtual ISR

Interrupt handling 2/2


```
void TIM1_UP_IRQHandler(void)
{
  Exec_IRQ_Handler(MC_IRQ_PWMNCURRFDBK_1,0);
}
```

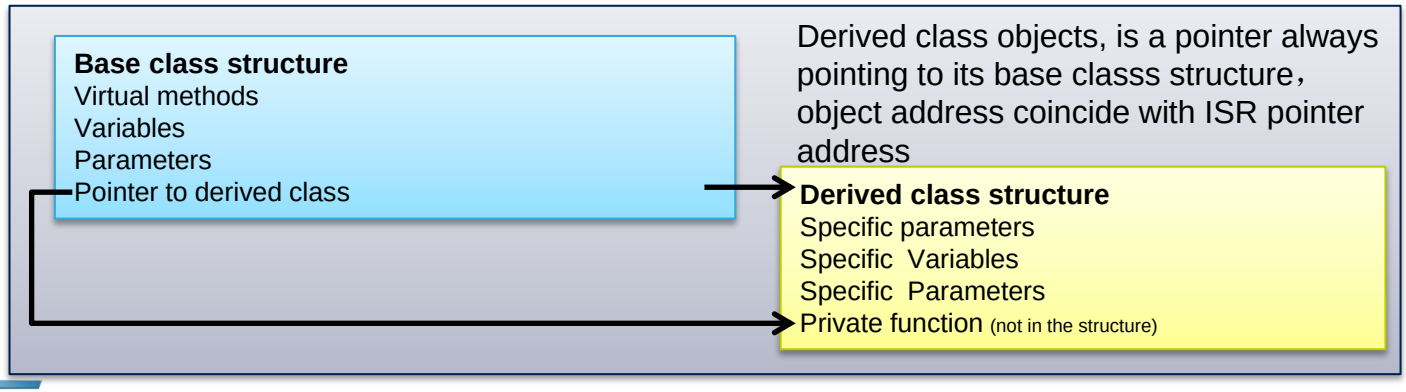
```
#define MC_IRQ_PWMNCURRFDBK_1  0u
#define MC_IRQ_PWMNCURRFDBK_2  1u
#define MC_IRQ_SPEEDNPOSFDBK_1  2u
#define MC_IRQ_SPEEDNPOSFDBK_2  3u
```

```
void Exec_IRQ_Handler(unsigned char bIRQAddr, unsigned char flag)
{
 oMC_IRQTable[bIRQAddr]->pIRQ_Handler((void*)(oMC_IRQTable)[bIRQAddr],flag);
}
```

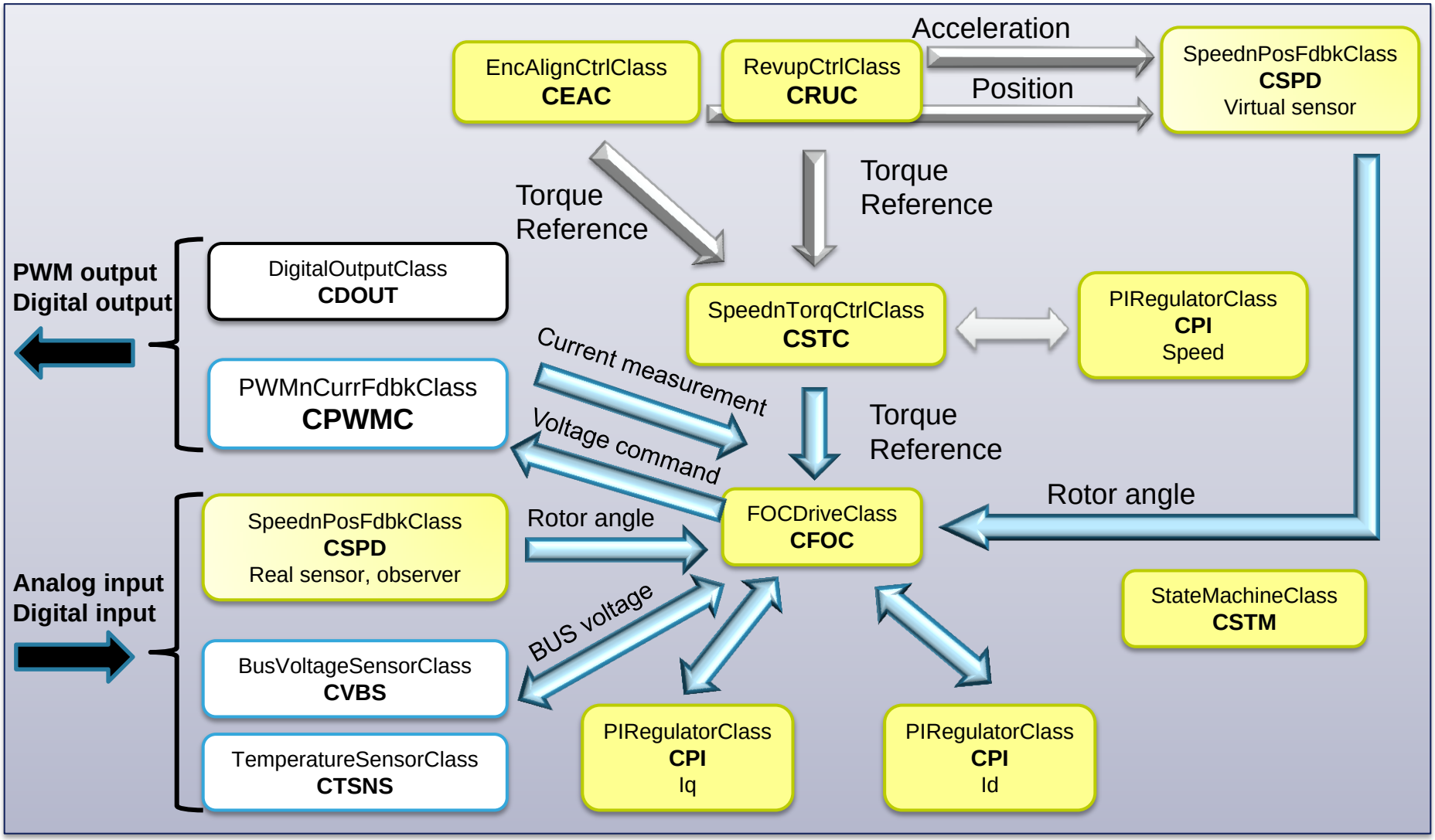
_CMCIRQ MC_IRQTable[MAX_MC_IRQ_NUM]

- oR1CurrentSensor
- oSpeedSensor
- …

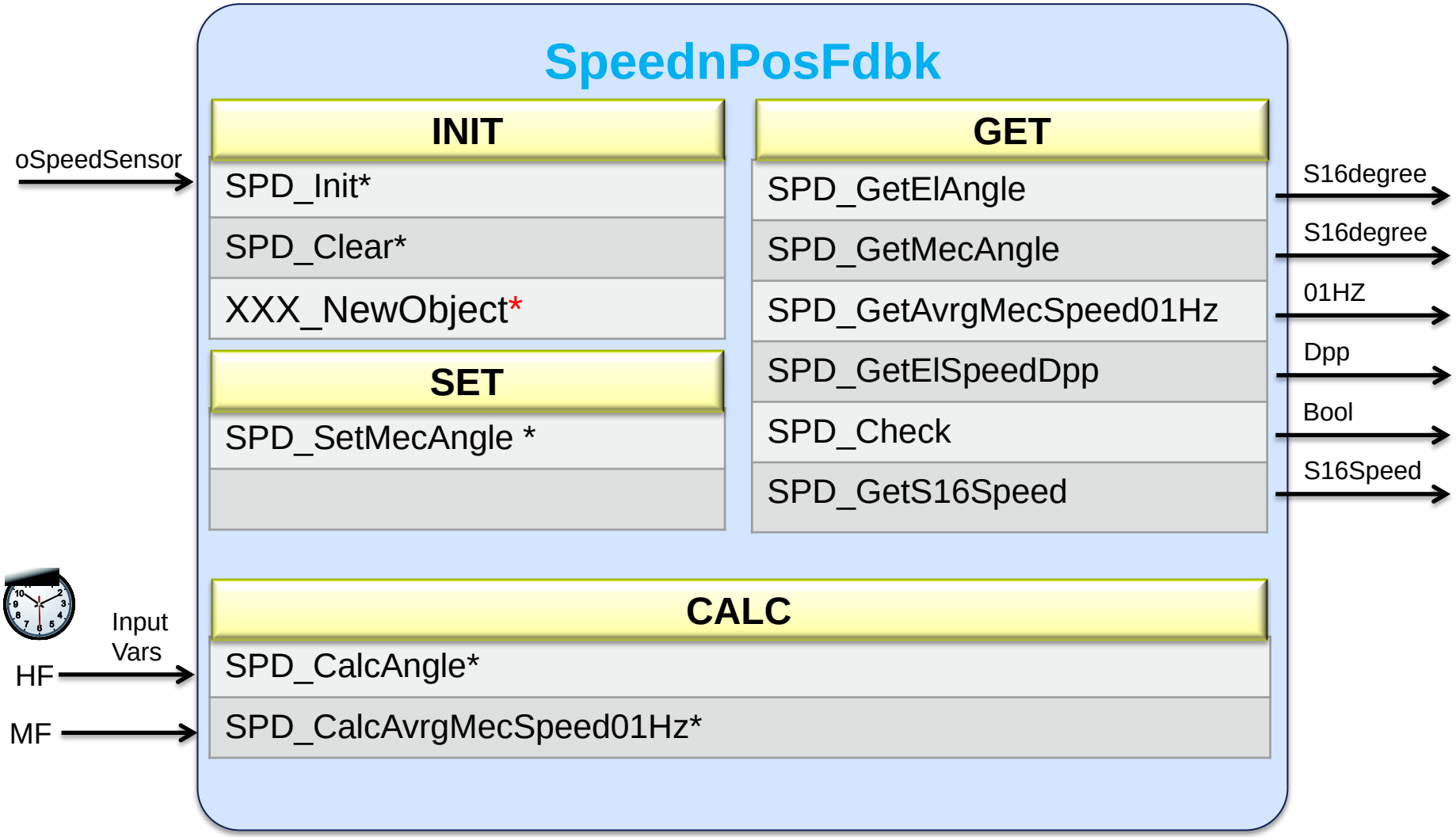
```
static void* R1HD2_IRQHandler(void *this, unsigned char flag)
{
 …
}
```



MC Library Information flow

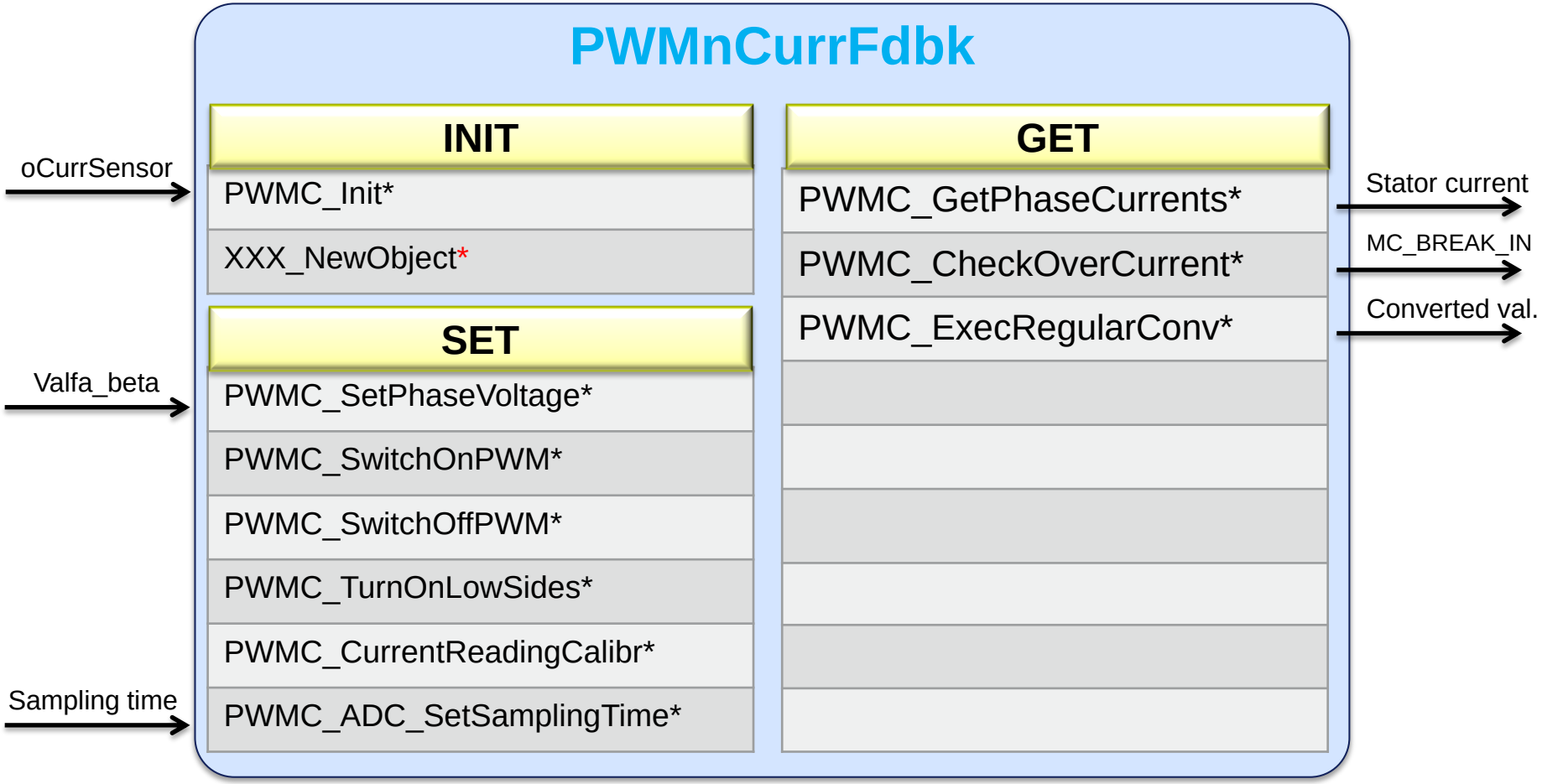


Speed and position feedback class



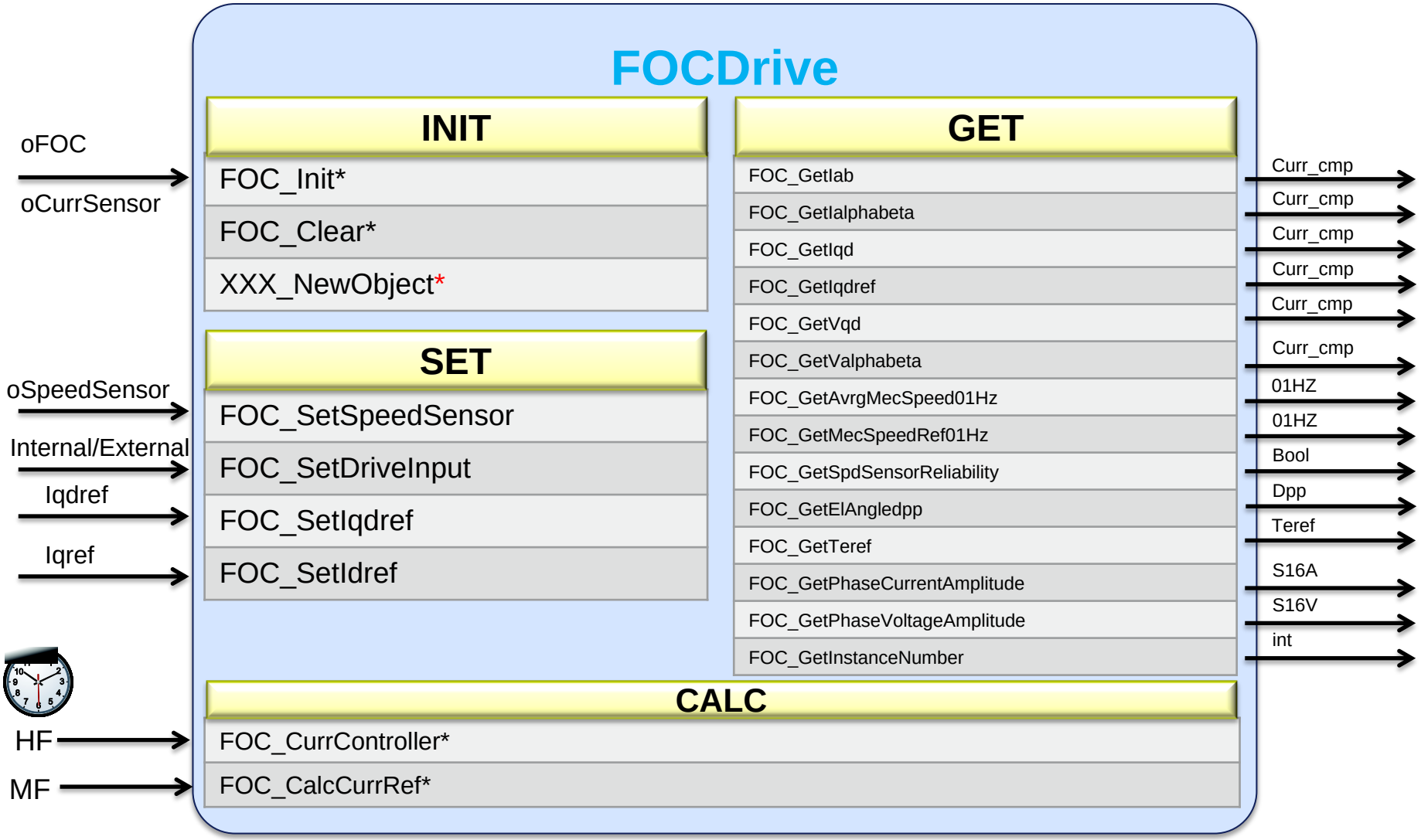
* XXX: Depend on the derived class

Current reading and PWM generation class



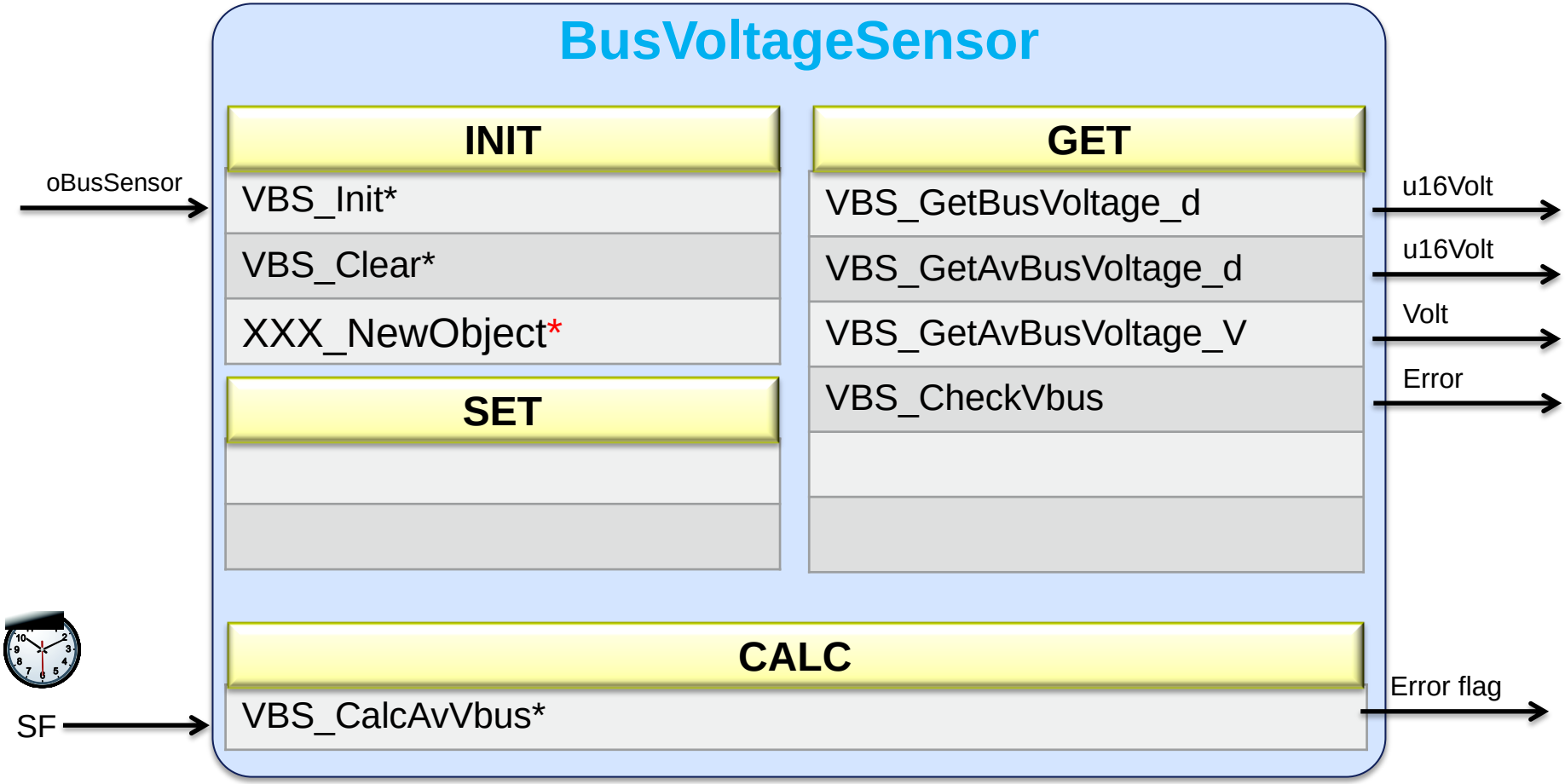
* XXX: Depend on the derived class

Field oriented control drive class



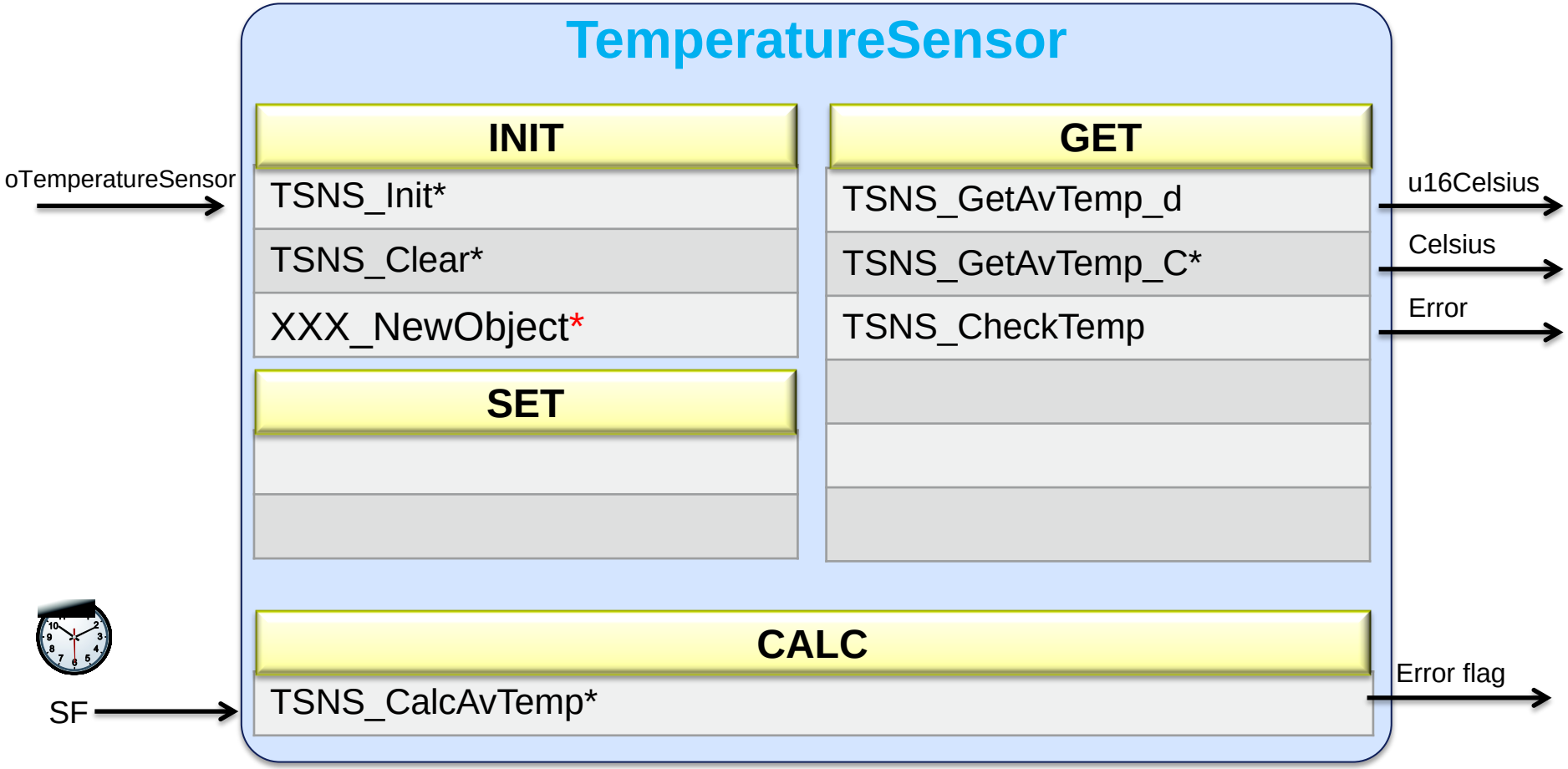
* XXX: Depend on the derived class

Bus voltage sensor class



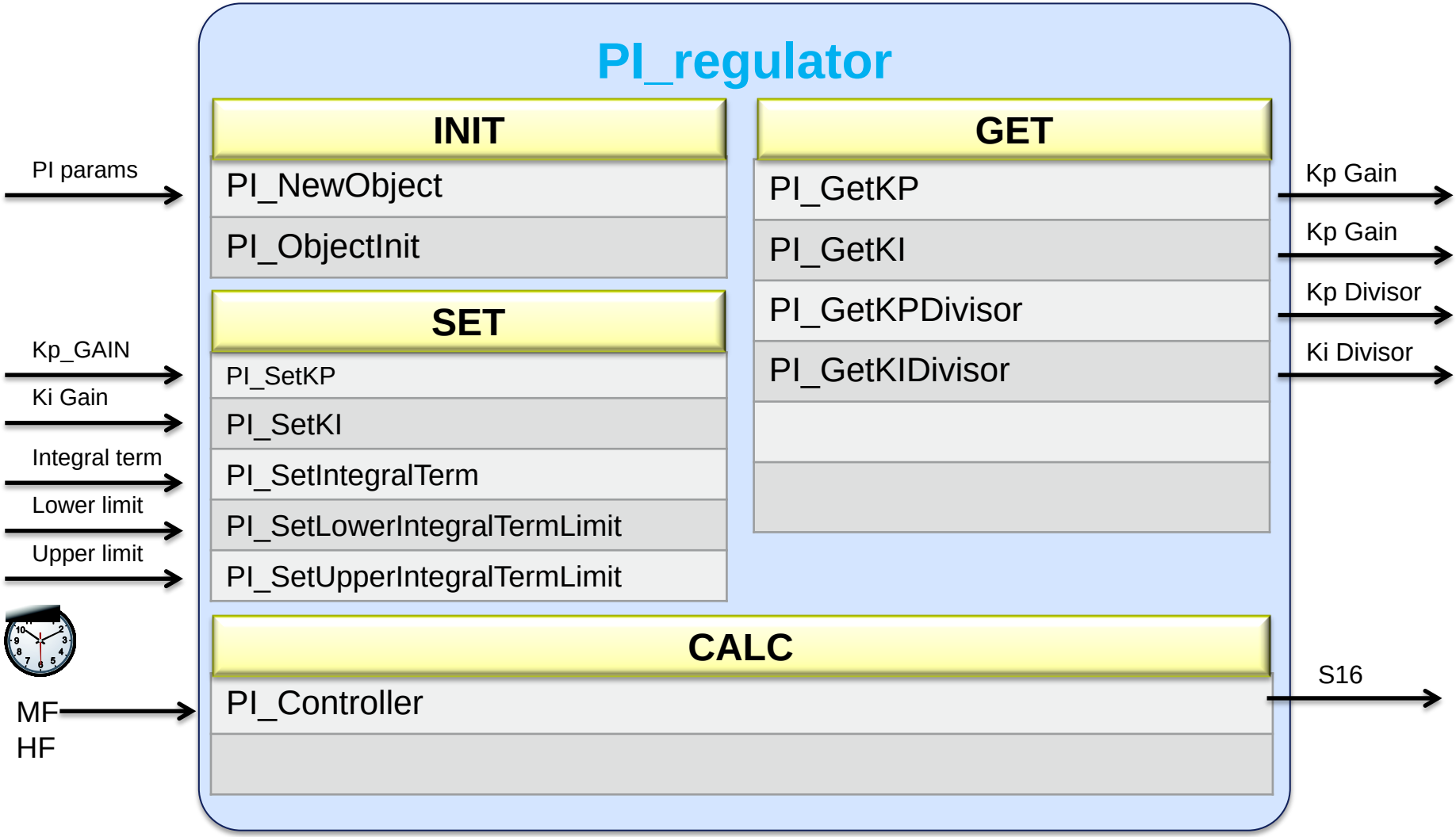
* XXX: Depend on the derived class(RVBS / VVBS)

Temperature sensor class

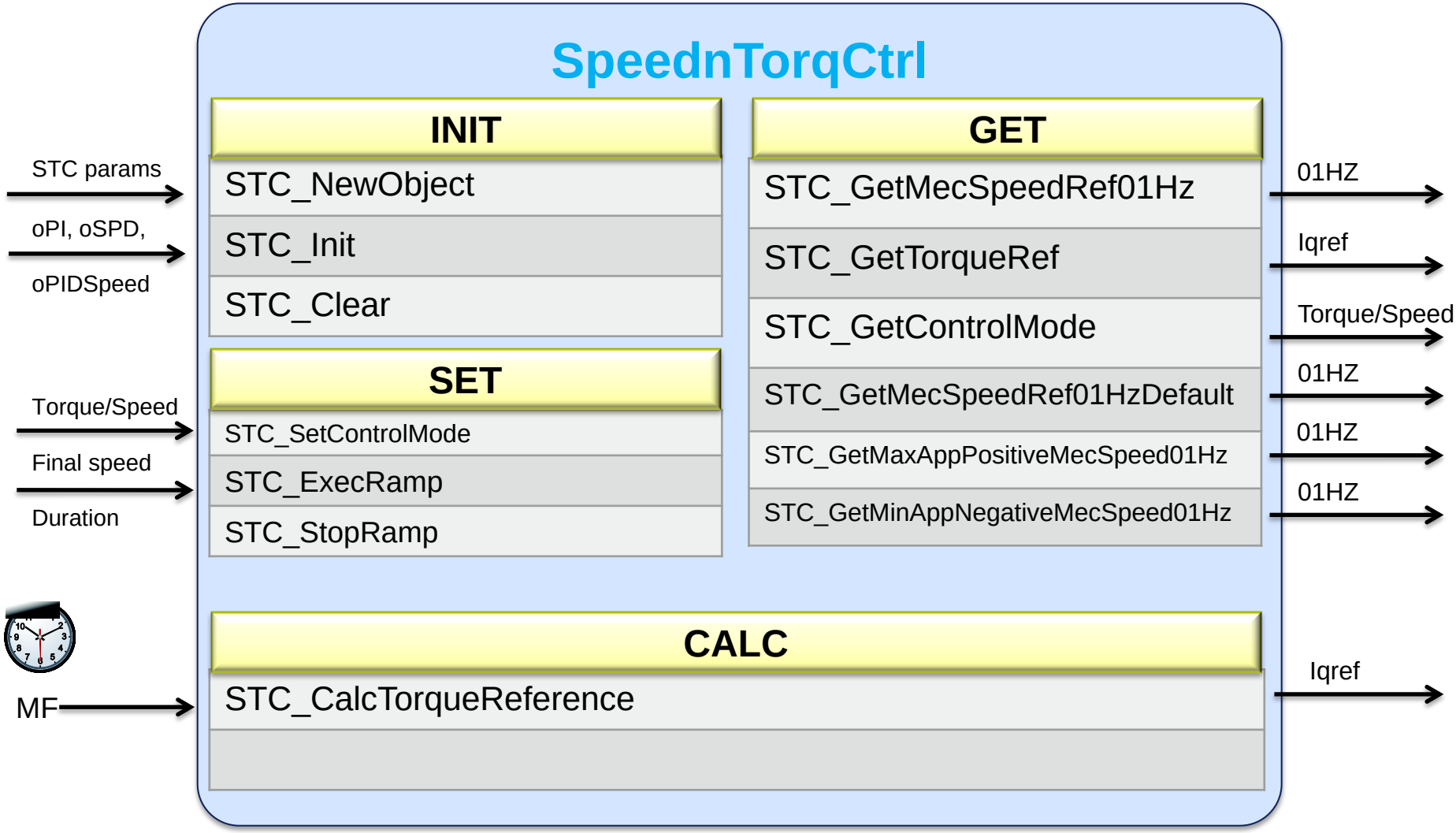


* XXX: Depend on the derived class(NTC / VTS)

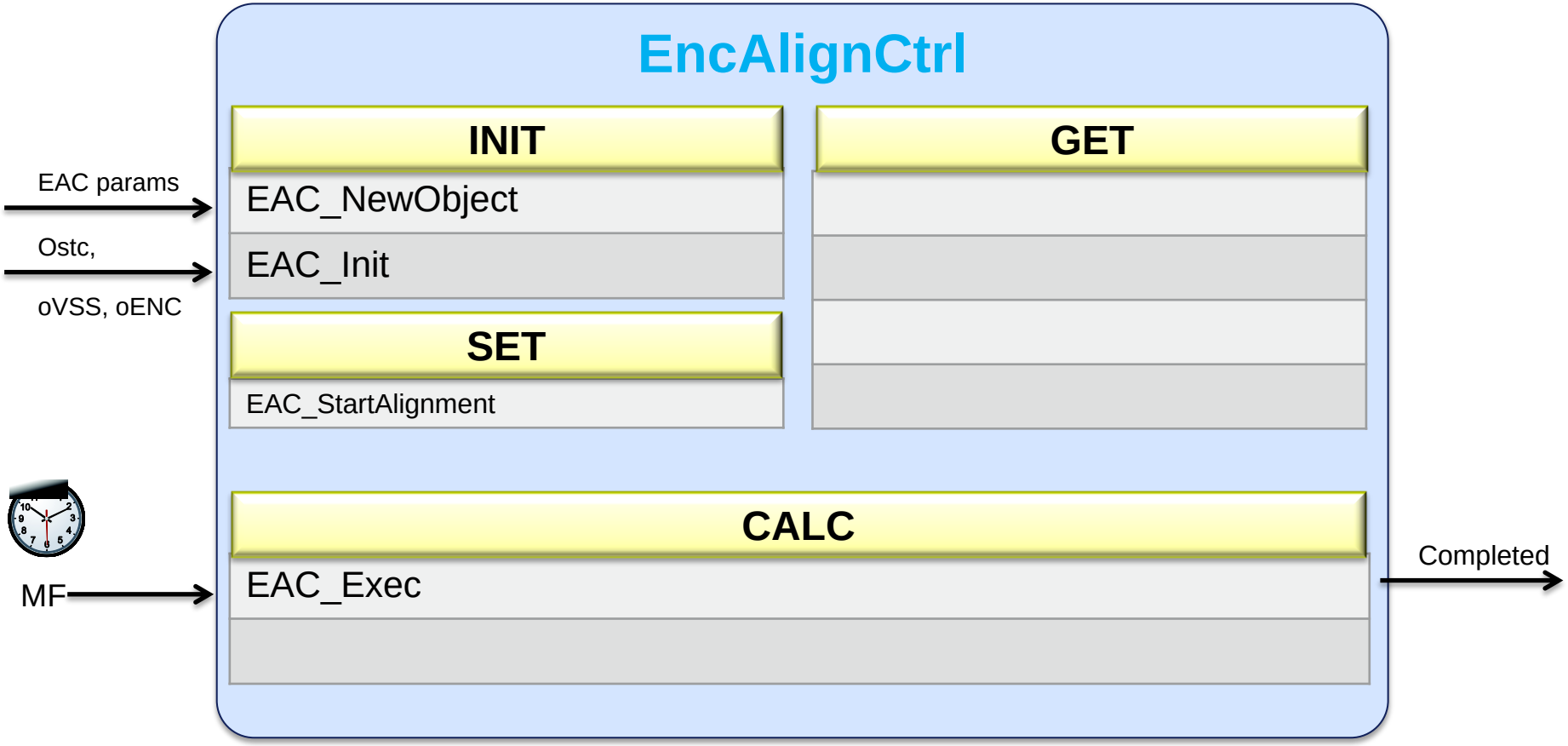
Proportional-integral regulator class



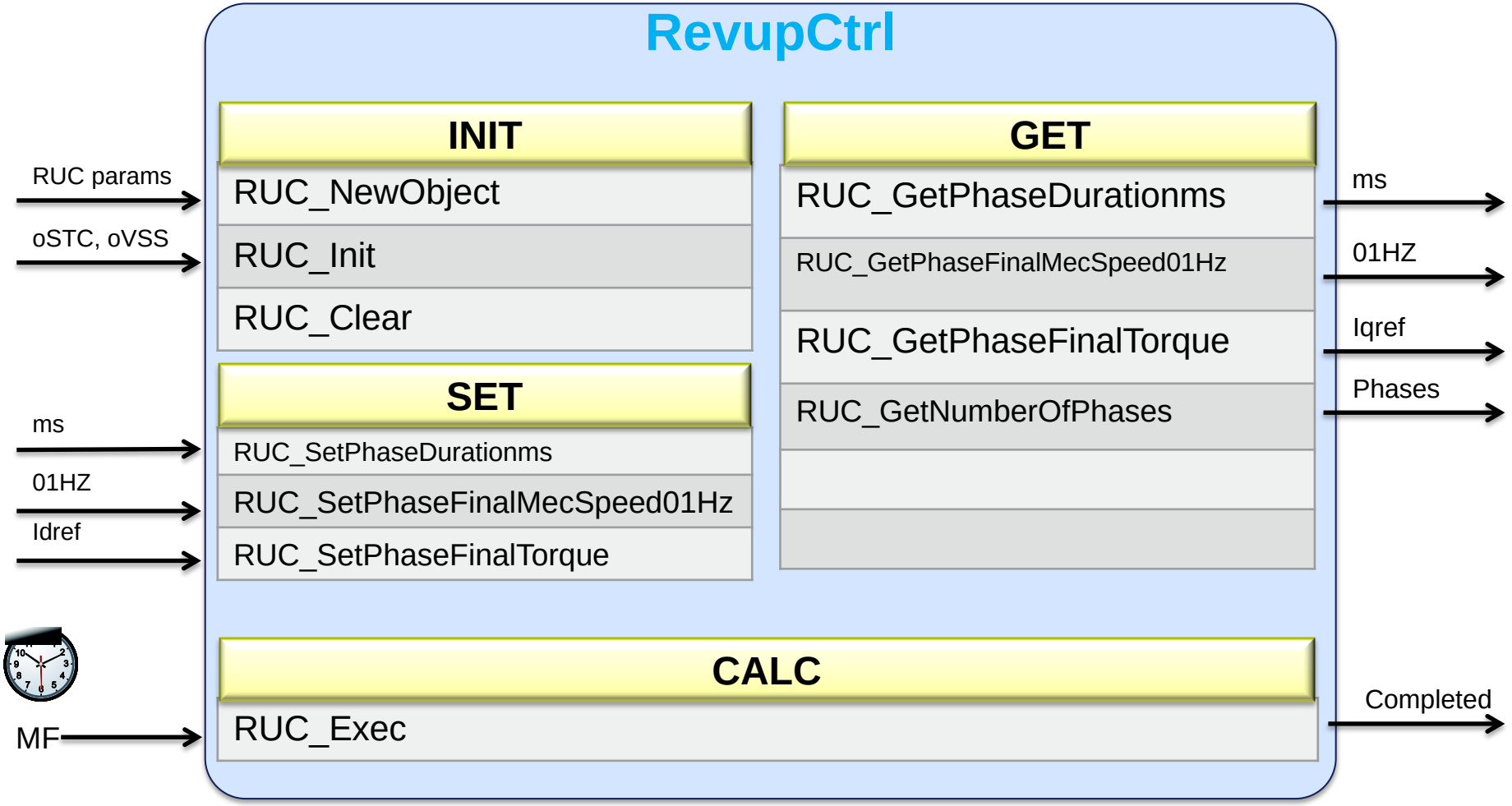
Speed and torque controller class



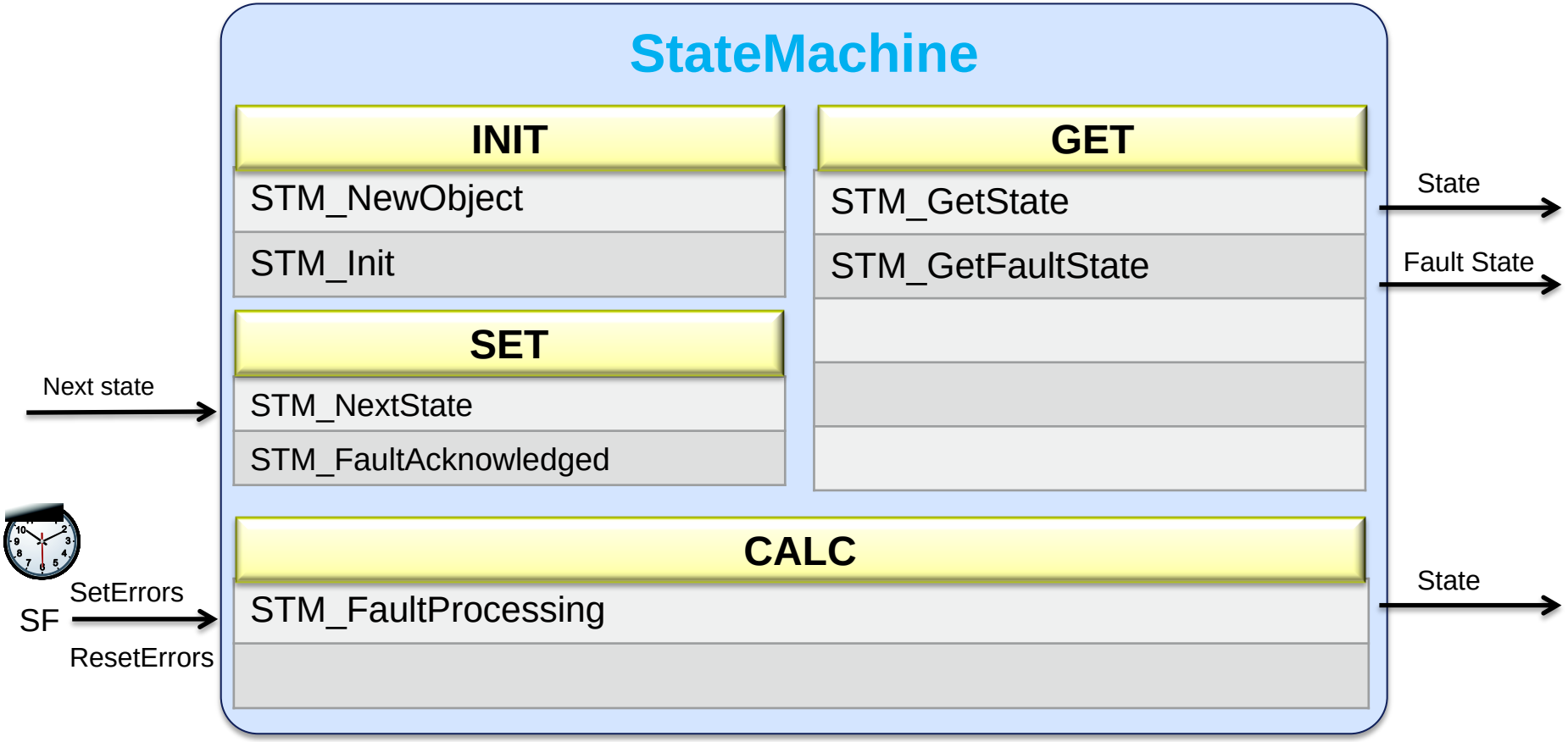
Encoder alignment controller class



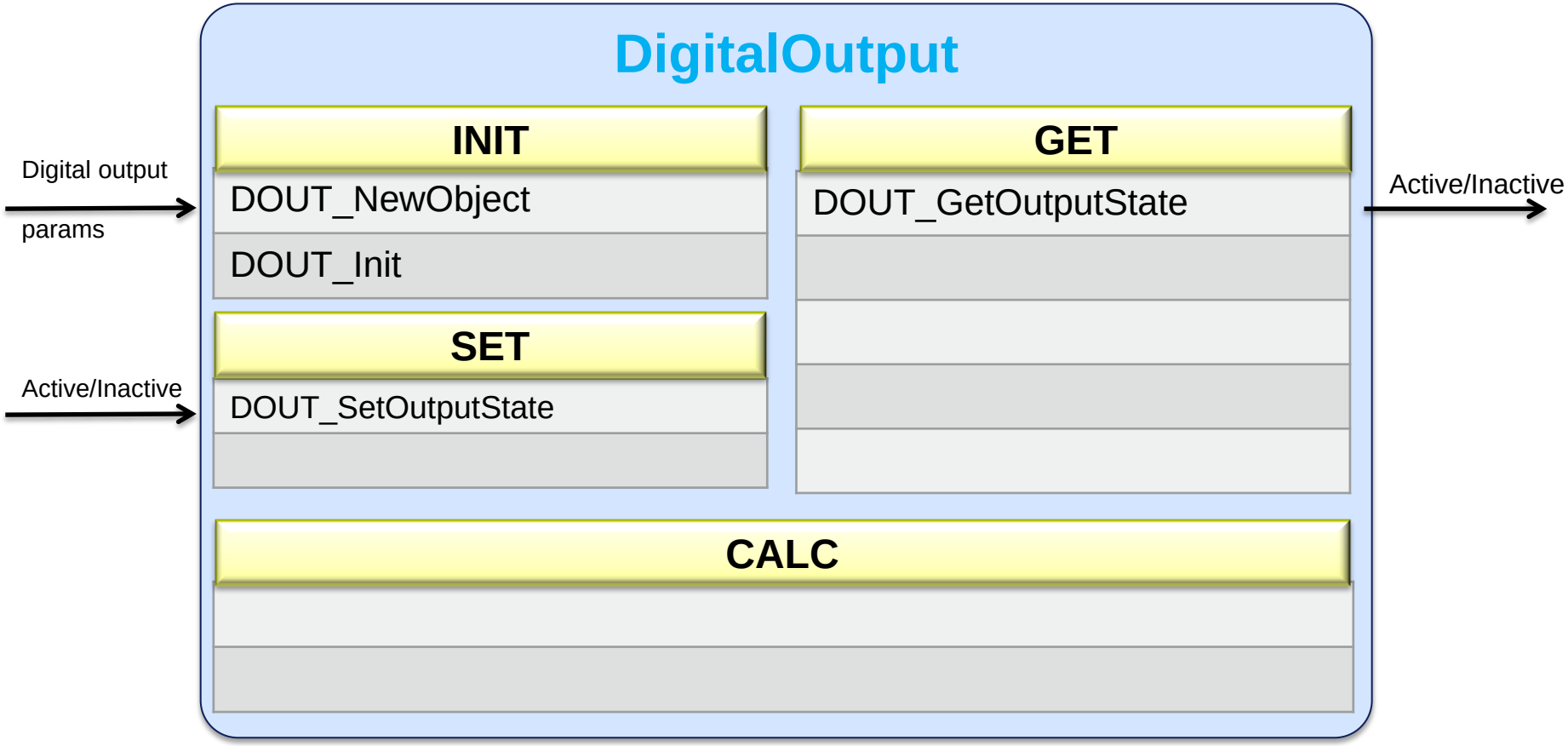
Rev-up controller class



State machine class



Digital output class



Help files

- Three help files is provided with the MC FOC library 3.2
- Help files directory: \install directory\Doc

File name	Description
STM32 FOC PMSM FW library v3_2 developer Help file.chm	This document aims to provide developer with a description of: • STM32 PMSM Motor control software library v3.2 classes interfaces. • STM32 PMSM Motor Control Interface API. • STM32 PMSM Motor control User Interface Library API.
STM32 PMSM MC Library v3.2 Developer manual.pdf	This document provides important information about the STM32 FOC PMSM FW library v3.2 with specific focus on its object-oriented programming implementation and its task-organized structure.
STM32 PMSM MC Library v3.2 User manual.pdf	This document aims to provide developer with a description of: • SDK constructure • Details of algorithm • API • Interface