User's Guide
SPRUIP4–May 2019

Fast Current Loop Driverlib Library

This user's guide provides a description of the fast current loop software library application program interface (API), which can be used for high-bandwidth, inner-loop control of AC servo drives with C2000 MCUs.

This document also explains the header files that are delivered with the library, and provides information on which CLA resources are used by the library and which PIE flags are cleared by the library.

Contents

1 Introduction 2
2 FCL Library Details 2
3 Building and Linking an Application With the Library 5

List of Tables

1 Summary of FCL APIs 2
2 Summary of Common Variables Across the Application and Library 3
3 Summary of CLA Resources Used by the Library 4

Trademarks

Code Composer Studio is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

1.1 Reference Example

Use this guide in conjunction with the example projects that use Fast Current Loop in the MotorControl SDK at:

C:\ti\c2000\C2000Ware_MotorControl_SDK_*version*\solutions\

The FCL software library can be found at:

C:\ti\c2000\C2000Ware_MotorControl_SDK_*version*\libraries\fcl\

2 FCL Library Details

2.1 API Overview

Table 1 lists the FCL APIs.

Table 1. Summary of FCL APIs

API Function	Description
uint32_t FCL_getSwVersion(void)	Returns a 32-bit constant; for this version the value returned is 0x00000006
void FCL_runComplexCtrl(void)	Performs the Complex control as part of the FCL
void FCL_runPICtrl(void)	Performs the PI control as part of the FCL
void FCL_runPICtrlWrap(void)	Wrap-up function called by the user application at the completion of the FCL in PI control mode
void FCL_runQEPWrap(void)	Called by the user application to handle the QEP feedback completion. This function is used only in FCL_LEVEL2.
void FCL_runComplexCtrlWrap(void)	Wrap-up function called by the user application at the completion of the FCL in Complex control mode
void FCL_initPWM(uint32_t basePhaseU, uint32_t basePhaseV, uint32_t basePhaseW)	Initializes PWMs for the FCL operation, this function is called by the user application during the initialization or setup process.
void FCL_resetController(void)	Called to reset the FCL variables and is useful when the user wants to stop and restart the motor.
void FCL_initQEP(uint32_t baseA)	This function initializes the eQEP peripheral for connecting to the QEP
void FCL_initADC(uint32_t resultBaseA, ADC_PPBNumber baseA_PPB, uint32_t resultBaseB, ADC_PPBNumber baseB_PPB, uint32_t adcBasePhaseW)	This function initializes the ADCs that are used to sense the motor phase currents

2.2 Header Files

2.2.1 Fast_Current_Loop.h

This header file contains general variables and pointers that are used across the application and the library.

Macro FCL_LIB is predefined when building the library and is not defined when the header file is included in the application. This helps applications use the same header file that is used by the library.

For example, in the following pointer declarations, when the header file is included in the library, the pointers are defined as extern, but when the same header file is included in the application the pointers are global. This helps the library work with variables that are common across the application and the software library.

```
#ifdef FCL_LIB
extern
#endif
```

```
CLARKE          clarke1;
```

This file also defines the following typedef of a structure used by the library, but the variables of the structure are initialized by the application, as shown in the provided example.

```
typedef struct _FCL_Parameters_ {
    float32_t  carrierMid,     // Mid point value of carrier count
               adcScale,       // ADC conversion scale to pu
               cmidsqrt3;      // internal variable

    float32_t
            tSamp,            // sampling time
            Rd,               // Motor resistance in D axis
            Rq,               // Motor resistance in Q axis
            Ld,               // Motor inductance in D axis
            Lq,               // Motor inductance in Q axis
            Vbase,            // Base voltage for the controller
            Ibase,            // Base current for the controller
            wccD,             // D axis current controller bandwidth
            wccQ,             // Q axis current controller bandwidth
            Vdcbus,           // DC bus voltage
            BemfK,            // Motor Bemf constant
            Wbase;            // Controller base frequency (Motor) in rad/sec
} FCL_Parameters_t;
```

Table 2 lists the variables needed by the library, which are supposed to be defined by the application. The same information is available in the Fast_Current_Loop.h header file delivered with the library. So it is sufficient if applications include the header file.

Table 2. Summary of Common Variables Across the Application and Library

Variable Name	Description or Use
extern uint16_t lsw;	Loop switch information controlled by both the library and the application
extern QEP qep1;	QEP feedback information accessed by both the application and the library
extern FCL_PI_CONTROLLER pi_iq;	PI IQ controller information accessed and handled by the CLA tasks and CPU inside the library and by CPU in the application
extern FCL_PI_CONTROLLER pi_id;	PI ID controller information accessed by both the library and the application
extern SVGEN svgen1;	Space Vector variables generated by the library are stored here.
extern RAMPGEN rg1;	Ramping up voltage vector angle, used during start up
extern SPEED_MEAS_QEP speed1;	Calculating speed from from QEP output
extern FCL_Parameters_t FCL_params;	Current Loop parameter constants that are to be initialized by the application. A reference function is provided in the example provided with the library.

2.2.2 fcl_pi.h

This file defines the following typedef of PI variables used in the library.

```
typedef struct {
   float32_t  ref;            // Input: reference set-point
   float32_t  fbk;            // Input: feedback
   float32_t  err;            // Output : error
   float32_t  out;            // Output: controller output
   float32_t  carryOver;      // Output : carrier over for next iteration
   float32_t  Kp;             // Parameter: proportional loop gain
   float32_t  Ki;             // Parameter: integral gain
   float32_t  Kerr;           // Parameter: gain for latest error
   float32_t  KerrOld;        // Parameter: gain for prev error
   float32_t  Umax;           // Parameter: upper saturation limit
   float32_t  Umin;           // Parameter: lower saturation limit             }
FCL_PIController_t;
```

2.3 CLA Resources Used

In this version of the library, the CLA resources in Table 3 are used and are unavailable for the user applications when using the provided software library.

Table 3. Summary of CLA Resources Used by the Library

FLC Controller	CLA Tasks Used
PI controller	CLA TASK1, CLA TASK2, and CLA TASK4
Complex controller	CLA TASK1, CLA TASK3, and CLA TASK4

2.3.1 CLA Task Prototypes

```
__interrupt void Cla1Task1();
__interrupt void Cla1Task2();
__interrupt void Cla1Task3();
__interrupt void Cla1Task4();
```

All the above tasks are declared and defined in the library. The assignment of the tasks to the appropriate CLA vectors is done in the user application.

The example provided with the library shows how to assign the tasks. The relevant code snippet is given below.

```
CLA_mapTaskVector(CLA1_BASE, CLA_MVECT_1, (uint16_t)(&Cla1Task1));
CLA_mapTaskVector(CLA1_BASE, CLA_MVECT_2, (uint16_t)(&Cla1Task2));
CLA_mapTaskVector(CLA1_BASE, CLA_MVECT_3, (uint16_t)(&Cla1Task3));
CLA_mapTaskVector(CLA1_BASE, CLA_MVECT_4, (uint16_t)(&Cla1Task4));
```

User applications are free to use the remaining CLA tasks, but these tasks are reserved according to Table 3, depending on the FCL controller option chosen.

2.4 Flags Cleared by the Library

Because the library uses the previously mentioned CLA tasks, it also clears the respective PIE IFR flag bits associated with the tasks.

2.5 Application Dependencies

The user application must initialize and clear the flags defined in this section for the library to be properly operational.

As shown in the example, all the parameters must be initialized before enabling any interrupts in the application initialization phase.

2.5.1 Initializing Current Loop Parameters for the Library

The following function, provided in the example code, initializes FCL_Pars. For more information, see Section 2.2.1 and Table 2.

```
initFCLVars();
```

2.5.2 Initializing PWM and PWM Access Pointers for the Library

The following code, shown in the example, initializes the PWM modules for the FCL library. This makes the library more portable, but it adds a slight cycle count during the execution of the library.

```
FCL_initPWM(EPWM1_BASE, EPWM2_BASE, EPWM3_BASE);
```

2.5.3 Initializing the ADC Int Flag and ADC PPB Result Register Pointers for the Library

The following code, shown in the example, initializes the ADC modules for Fast control loop library. This makes the library more portable but adds a slight cycle count during the execution of library.

```
FCL_initADC (ADCARESULT_BASE, ADC_PPB_NUMBER1,
             ADCBRESULT_BASE, ADC_PPB_NUMBER1,
             ADCA_BASE);
```

2.5.4 Initializing the EQEP Access Pointer for the Library

The following code, shown in the example, initializes the EQEP registers pointer for the library to access.

```
FCL_initQEP(EQEP1_BASE);
```

2.5.5 Configuring and Clearing the CLA TASK1 Trigger

User applications, as shown in the provided example, must be configured to trigger the CLA TASK1 by the same event that triggers the ADC SOC.

The following code in the example shows the initialization of CLA and setting up the CLA TASK1 trigger. This must be performed before enabling the PWM clocks.

```
//initialize CLA for FCL library
configureCLA();

//Enable EPWM1 INT trigger for CLA TASK1
CLA_setTriggerSource(CLA_TASK_1, CLA_TRIGGER_EPWM1INT);
```

Similarly, the user application must also clear the event that triggers the CLA task in the user code. This is also shown in the example provided with the library.

```
EPWM_clearEventTriggerInterruptFlag(EPWM1_BASE);
```

3 Building and Linking an Application With the Library

The example project(s) using the library demonstrates integrating this library into an application running from flash/RAM.

The appropriate linker command files are also provided with the example project(s).Because the library uses CLA, RAM must be shared across the CPU and CLA. The example project(s) shows how to do this as well.

The library file '*fcl_cpu_cla.lib*' is an index library that selects between a COFF ABI format or an EABI format library. The settings of the example project determines the choice of format to use.

The library is built with compiler version v18.12.1 LTS using Code Composer Studio™ v9 IDE.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265