Kinetis Program for 32-bit Flash Memory Programming

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<stdint.h>
#include<string.h>
#include<math.h>

#define F_OFFSET (4002 << 16)
#define F_STAT 0
#define F_CNFG 1
#define F_FSEC 2
#define F_FOPT 3
#define FFCOB3 4
#define FFCOB2 5
#define FFCOB1 6
#define FFCOB0 7
#define FFCOB7 8
#define FFCOB6 9
#define FFCOB5 A
#define FFCOB4 B
#define FFCOBB C
#define FFCOBA D
#define FFCOB9 E
#define FFCOB8 F

#define F_STAT_CCIF_POSITION 7
#define F_STAT_RDCOLERR_POSITION 6
#define F_STAT_ACCERR_POSITION 5
#define F_STAT_FPVIOL_POSITION 4
#define F_STAT_MGSTAT0_POSITION 0

#define FCMD_READ_SEC 0X01
#define FCMD_PROG_CHK 0X02
#define FCMD_READ_RSC 0X03
#define FCMD_PROG_LGW 0X06
#define FCMD_ERSE_SEC 0X09
#define FCMD_READ_BLK 0X40
#define FCMD_READ_ONC 0X41
#define FCMD_PROG_ONC 0X43
#define FCMD_ERSE_ALL 0X44
#define FCMD_BKDR_KEY 0X45

#define SECTOR_SIZE 1024
#define FLASH_START_ADDRESS 0X400
#define FLASH_END_ADDRESS 0X20000

#define REG_WRITE(address, value)       (*(vuint8_t*)(address) = (value))

typedef enum {
    /* add error codes here as needed */
    F_CCIF   = -3  /* operation flag - shows 0 if the flash is busy */
    F_ACCERR = -2, /* attempt to read while program in progress */
    F_FPVIOL = -1, /* attempt to program a protected flash area */
    F_NO_ERROR = 0 /* success */
} ferr_t;
```

```
ferr_t fprogram( uint32_t *address, uint32_t data );

// Function to erase a sector

/* Flushing (erasing) sector values that the address is contained in: */

	uint8_t Flash_Sector_Erase (vuint32_t *Address)
	{

// Clearing the specified sector_address value
	uint32_t sector_address = 0;

// Analysis to see if the address is contained within the allowed memory
boundaries

	if((*Address >= FLASH_START_ADDRESS) && (*Address < FLASH_END_ADDRESS))

	{
		// Finding the beginning address of the sector from addresses
available
		sector_address = ((uint8_t)(*Address / Sector_Size)) *
Sector_Size;
		printf("\nThe sector that is in use is %d for which we perform our
Erase Operation\n", sector_address);
	}
	else
	{
		printf("\nUnfortunately, the address is out of order!\n");
	}

//  Error Checking for CCIF

	F_CCIF_WAIT_STATE(); // Wait for CCIF to be set to 1

	//Checking and clearing errors for previous command execution
	uint8_t ERROR; // declaring the ERROR flag
	ERROR = F_CHK_ERROR(); //Checking if any error
	if(ERROR)
	{
		F_CLR_ERROR(); // Error clearing macro
	}

	//Invoking parameters
	REG_WRITE((F_OFFSET + FCCOB0), \
				(vuint8_t)FCMD_Erase_Sector);

	REG_WRITE((F_OFFSET + FCCOB1), \
				(vuint8_t)(GET_BIT_16_23(sector_address)));

	REG_WRITE((FTFA_Address_Offset + FCCOB2), \
				(vuint8_t)(GET_BIT_8_15(sector_address)));

	REG_WRITE((F_OFFSET + FCCOB3), \
				(vuint8_t)(GET_BIT_0_7(sector_address)));

	//Launching command and waiting for it to finish
```

```
	F_EXE_OPERATION(); // Function to execute the flash code operation
	F_CCIF_WAIT_TAG(); // Wait for CCIF to be set to 1

	//If ERROR engendered, the following case will be considered:
	ERROR = 0;   // When error is detected, ERROR turns to 0
	ERROR = F_CHK_ERROR(); // The flash ERROR check function is executed
	return (ERROR); // ERROR is returned by the function call
}


//Flash code for 32-bit programming
int8_t F_LONGWORD_WR(vuint32_t *Address, vuint32_t Data)
{
	//Balance check
	if(F_LONGWORD_WR_ADD_BALANCE(*Address))
return (F_LONGWORD_BALANCE_ERROR);
	//Boundary check for the specified address if it falls within the
specified memory bandwidth
	if((*Address < FLASH_START_ADDRESS) || (*Address >= FLASH_END_ADDRESS))

{
	printf("\nUnfortunately, the address is out of order!\n");
}

	F_CCIF_WAIT_STATE(); // Wait for CCIF to be set to 1
	uint8_t ERROR; // declaring the ERROR flag
	ERROR = F_CHK_ERROR(); //Checking if any error
	if(ERROR)
	{
		F_CLR_ERROR(); // Error clearing macro
	}

	REG_WRITE((F_OFFSET + FCCOB0), \
				(vuint8_t)FCMD_Program_Longword);

	REG_WRITE((F_OFFSET + FCCOB1), \
				(vuint8_t)(GET_BIT_16_23(*Address)));

	REG_WRITE((F_OFFSET + FCCOB2), \
				(vuint8_t)(GET_BIT_8_15(*Address)));

	REG_WRITE((F_OFFSET + FCCOB3), \
				(vuint8_t)(GET_BIT_0_7(*Address)));

	REG_WRITE((F_OFFSET + FCCOB4), \
				(vuint8_t)(GET_BIT_0_7(Data)));

	REG_WRITE((F_OFFSET + FCCOB5), \
				(vuint8_t)(GET_BIT_8_15(Data)));

	REG_WRITE((F_OFFSET + FCCOB6), \
				(vuint8_t)(GET_BIT_16_23(Data)));

	REG_WRITE((F_OFFSET + FCCOB7), \
				(vuint8_t)(GET_BIT_24_31(Data)));

	//Launching command and waiting for it to finish
```

```
        F_EXE_OPERATION(); // Function to execute the flash code operation
        F_CCIF_WAIT_TAG(); // Wait for CCIF to be set to 1


        //If ERROR engendered;
        ERROR = 0; // When error is detected, ERROR turns to 0
        ERROR = F_CHK_ERROR(); // The flash ERROR check function is executed
        return (ERROR); // ERROR is returned by the function call
}
```

Alternate C++ code for brief explanation of the style of execution of the above Embedded C code

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<stdint.h>
#include<string.h>
#include<math.h>

// Flash Memory Programming

/* Global Variables */

uint32_t *address;
uint32_t data;
uint32_t tst_address;
uint8_t tst, read_write, output;

/* Error States */

uint8_t CCIF = 1;

/* Error Description */

typedef enum func_errors
{
   /* Implemented Error Codes */
   F_NO_ERROR          = 0, /* success */
        F_ACCERR       = 1, /* attempt to read while program in progress */
   F_FPVIOL      = 2, /* attempt to program a protected flash area */
   F_CCIF              = 3  /* operation flag - shows 0 if the flash is busy */
} ferr_t;

/* Functions introduced */
```

```
ferr_t fprogram( uint32_t *address, uint32_t data );

/* Main Function */

main()
{
	while(1)
	{
		printf("\nWelcome to this platform on Flash Programming\n");
		printf("If you wish to perform flash-programming for a 32-bit word, please type 1 (yes) or enter any other value to exit the program:\n");
		scanf("%d", &tst);
		while (tst == 1)
		{
			if (CCIF == 1)
			{
				fprogram(address, data);
				printf("output: %d", output);
				printf("\nThank you for using our services\n");
			}
			else
			{
				printf("\nAn ERROR is DETECTED...\n");
				output = F_CCIF;
				printf("output: %d", output);
				printf("\nYou have successfully terminated the program execution. We will revert you back to the execution starting point\n\n\n");
			}
			return 0;
		}
		printf("\nYou have successfully terminated the program execution. We will revert you back to the execution starting point\n\n\n");
	}
	return 0;
}

/* Function Definitions */

ferr_t fprogram( uint32_t *address, uint32_t data )
{
		printf("\nIf you wish to read a memory location, enter 1 or else enter anything for writing operation:\n");
		scanf("%d", &read_write);
		if (read_write == 1)
		{
			printf("\nPlease, enter the absolute address in HEX of the flash memory location from where you wish to READ a 32-bit data\n");
			scanf("%x", &tst_address);
			if(tst_address >= 0x400 && tst_address <= 0x20000)
			{
```

```
//                              uint32_t *address = &tst_address;
//                              *address = 0x00000000;
                                printf("\nThe data read at the specified memory location: %x is :
%x", tst_address, data);
                                printf("Operation SUCCESSFUL - NO ERRORS");
                                output = F_NO_ERROR;
                        }
                        else
                        {
                                printf("\nAn ERROR is DETECTED...\n");
                                output = F_FPVIOL;
                        }
                }
                else
                {
                        printf("\nPlease, enter the absolute address in HEX of the flash memory
location where you wish to write your 32-bit data\n");
                        scanf("%d", &tst_address);
                        if(tst_address >= 0x400 && tst_address <= 0x20000)
                        {
                                printf("Please enter your 32-bit data in hex\n");
                                scanf("%x", &data);
//                              uint32_t *address = &tst_address;
//                              *address = data;
                                printf("\nData written at the specified memory location: %x is
%x", tst_address, data);
                                printf("Operation SUCCESSFUL - NO ERRORS");
                                output = F_NO_ERROR;
                        }
                        else
                        {
                                printf("\nAn error is detected...\n");
                                output = F_FPVIOL;
                        }
                }
}
```