Class instances;

cL6470();
In this instance you do not need to pass anyparameter.just create the instance and after that assign the Pin (for assigning the Pin please check the x_Set_x function, x means CS, MOIS,MISO,Reset,Busy)
Example cL6470 Motor();

cL6470(MOSI, MISO, CS, CLK, BUSY , RESET);					
Param1 Type Int MOSI Pin	Param2 Type Int MISO Pin	Param3 Type Int CS Pin	Param4 Type Int CLK Pin	Param5 Type Int BUSY Pin	Param6 Type Int RESET Pin
Example cL6470 Motor(11,12,10,13,5,6);					

Pin Assigning Functions

if you create the instance without parameters then you can use these function to assign the Pin Number to the instance.

x_Set_MOSI(MOSI_Pin_Number);
Param Type Integer MOSI Pin Number
Example Motor. x_Set_MOSI(11);

x_Set_MISO (MISO_Pin_Number);
Param Type Integer MISO Pin Number
Example Motor. x_Set_MISO(12);

x_Set_CS (CS_Pin_Number);
Param Type Integer CS Pin Number
Example Motor. x_Set_CS(10);
x_Set_CLK (CLK_Pin_Number);
Param Type Integer CLK Pin Number
Example Motor. x_Set_CLK(13);

x_Set_BUSY (BUSY_Pin_Number);
Param Type Integer BUSY Pin Number
Example Motor. x_Set_BUSY(5);

x_Set_RESET(RESET_Pin_Number);
Param Type Integer RESET Pin Number
Example Motor. x_Set_RESET(6);

You can also Get the PiNumber.Here the Some Function to get the Pin Number

- _Get_MOSI();
- _Get_MISO();
- _Get_CS();
- _Get_CLK();
- _Get_BUSY();
- _Get_RESET();

return Type is Integer.

initialization

so first thing come first.please initialization the pins. for this we design the function.

please use initialization in the Setp of the code. for more understanding please check the example .

init();
He has no Parameters
Example Motor. init();

After the initialization, You need to adjust the some parameters for the Motor, to adjust the parameters we have two functions.please Check the L6470 DataSheet for defination of these parameters. P40

Parameters Micros

- x_ABS_POS
- x_EL_POS
- x_MARK
- x_SPEED
- x_ACC
- x_DEC
- x_MAX_SPEED
- x_MIN_SPEED
- x_FS_SPD
- x_KVAL_HOLD
- x_KVAL_RUN
- x_KVAL_ACC
- x_KVAL_DEC
- x_INT_SPD
- x_ST_SLP
- x_FN_SLP_ACC
- x_FN_SLP_DEC
- x_K_THERM
- x_ADC_OUT
- x_OCD_TH
- x_STALL_TH
- x_STEP_MODE
- x_ALARM_EN
- x_CONF
- x_STATUS

_GetParam(ParametersMicros)
Param Type Byte Please use the above define Micros. to get the parameters value.the return type is Byte.
Example Motor_GetParam(x_CONFIG) == 0x2E88

_SetParam(ParametersMicros, Value);	
Param1 Type Byte Please use the above defined micros.	Param2 Type unsigned Long Value for this Specified Parameters.
Example Motor._SetParam(x_INT_SPD, 0x3550);	

When you are Setting the over current thresholds parameters (x_OCD_TH). we have some predefoned Micros.

- x_OCD_TH_375mA
- x_OCD_TH_750mA
- x_OCD_TH_1125mA
- x_OCD_TH_1500mA
- x_OCD_TH_1875mA
- x_OCD_TH_2250mA
- x_OCD_TH_2625mA
- x_OCD_TH_3000mA
- x_OCD_TH_3375mA
- x_OCD_TH_3750mA
- x_OCD_TH_4125mA
- x_OCD_TH_4500mA
- x_OCD_TH_4875mA
- x_OCD_TH_5250mA
- x_OCD_TH_5625mA
- x_OCD_TH_6000mA

For example

Motor._SetParam(x_OCD_TH, x_OCD_TH_6000mA);

When you are Setting the StepMode parameters (x_STEP_MODE).we have some predefoned Micros.

- x_STEP_SEL_1

- x_STEP_SEL_1_2
- x_STEP_SEL_1_4
- x_STEP_SEL_1_8
- x_STEP_SEL_1_16
- x_STEP_SEL_1_32
- x_STEP_SEL_1_64
- x_STEP_SEL_1_128

For example

```
Motor._SetParam(x_STEP_MODE,
                !x_SYNC_EN |x_STEP_SEL_1_128 |
                x_SYNC_SEL_1);
```

x_SYNC_EN controls whether the BUSY/SYNC pin reflects the step frequency or the BUSY status of the chip. We want it to be the BUSY status. so we pass the inverted Value.

x_STEP_SEL_x is the microstepping rate- we'll go full step 128.

x_SYNC_SEL_x is the ratio of (micro)steps to toggles on the BUSY/SYNC pin (when that pin is used for SYNC). but we are using Busy because we connect the Blue light to this Pin.

Some Predefined Micros for the configuration Parameters.

Oscillator options. The x needs to know what the clock frequency is because it uses that for some calculations during operation.

x_CONFIG_OSC_SEL	Mask for this bit field.
x_CONFIG_INT_16MHZ	Internal 16MHz, no output
x_CONFIG_INT_16MHZ_OSCOUT_2MHZ	Default; internal 16MHz, 2MHz output
x_CONFIG_INT_16MHZ_OSCOUT_4MHZ	Internal 16MHz, 4MHz output
x_CONFIG_INT_16MHZ_OSCOUT_8MHZ	Internal 16MHz, 8MHz output
x_CONFIG_INT_16MHZ_OSCOUT_16MHZ	Internal 16MHz, 16MHz output
x_CONFIG_EXT_8MHZ_XTAL_DRIVE	External 8MHz crystal
x_CONFIG_EXT_16MHZ_XTAL_DRIVE	External 16MHz crystal
x_CONFIG_EXT_24MHZ_XTAL_DRIVE	External 24MHz crystal
x_CONFIG_EXT_32MHZ_XTAL_DRIVE	External 32MHz crystal
x_CONFIG_EXT_8MHZ_OSCOUT_INVERT	External 8MHz crystal, output inverted
x_CONFIG_EXT_16MHZ_OSCOUT_INVERT	External 16MHz crystal, output inverted

x_CONFIG_EXT_24MHZ_OSCOUT_INVERT External 24MHz crystal, output inverted
x_CONFIG_EXT_32MHZ_OSCOUT_INVERT External 32MHz crystal, output inverted

Configure the functionality of the external switch input
x_CONFIG_SW_HARD_STOP Default; hard stop motor on switch.
x_CONFIG_SW_USER Tie to the GoUntil and ReleaseSW

Configure the motor voltage compensation mode (see page 34 of datasheet)
x_CONFIG_VS_COMP_DISABLE Disable motor voltage compensation
x_CONFIG_VS_COMP_ENABLE Enable motor voltage compensation

Configure overcurrent detection event handling
x_CONFIG_OC_SD_DISABLE Bridges do NOT shutdown on OC detect
x_CONFIG_OC_SD_ENABLE Bridges shutdown on OC detect

Configure the slew rate of the power bridge output
x_CONFIG_SR_180V_us
x_CONFIG_SR_290V_us
x_CONFIG_SR_530V_us

Integer divisors for PWM sinewave generation
See page 32 of the datasheet for more information on this.
x_CONFIG_PWM_MUL_0_625
x_CONFIG_PWM_MUL_0_75
x_CONFIG_PWM_MUL_0_875
x_CONFIG_PWM_MUL_1
x_CONFIG_PWM_MUL_1_25
x_CONFIG_PWM_MUL_1_5
x_CONFIG_PWM_MUL_1_75
x_CONFIG_PWM_MUL_2

Multiplier for the PWM sinewave frequency
x_CONFIG_PWM_DIV_1
x_CONFIG_PWM_DIV_2
x_CONFIG_PWM_DIV_3
x_CONFIG_PWM_DIV_4
x_CONFIG_PWM_DIV_5
x_CONFIG_PWM_DIV_6
x_CONFIG_PWM_DIV_7

Example

```
Motor._SetParam(x_CONFIG,
                x_CONFIG_PWM_DIV_1 |
                x_CONFIG_PWM_MUL_2 |
                x_CONFIG_SR_180V_us |
                x_CONFIG_OC_SD_DISABLE |
                x_CONFIG_VS_COMP_DISABLE |
                x_CONFIG_SW_HARD_STOP |
                x_CONFIG_INT_16MHZ);
```

Functions

Fetch and return the 16-bit value in the STATUS register. Resets any warning flags and exits any error states. return Type is integer.
_GetStatus();

This is a 20-bit value, so we need to make sure the value is at or below 0xFFFFF. return type is unsigned Long.
_SpdCalc(float);

This is a 14-bit value, so we need to make sure the value is at or below 0x3FFF. return type is unsigned Long.
_IntSpdCalc(float);

_FSCalc contains a threshold value above which microstepping is disabled return type is unsigned Long.
_FSCalc(float);

Adjust the minimum Speed. This is a 12-bit value, so we need to make sure the value is at or below 0xFFF. return type is unsigned Long.
_MinSpdCalc(float);

Adjust the maximum Speed. This is a 10-bit value, so we need to make sure it remains at or below 0x3FF. return type is unsigned Long.
_MaxSpdCalc(float);

The calculation for DEC is the same as for ACC. Value is 0x08A on boot.This is a 12-bit value, so we need to make sure the value is at or below 0xFFF. return type is unsigned Long.
_DecCalc(float);
_AccCalc(float);

Enable or disable the low-speed optimization option.
_SetLSPDOpt(boolean);

RUN sets the motor spinning in a direction (defined by the constants.FWD and REV). Maximum speed and minimum speed are defined by the MAX_SPEED and MIN_SPEED registers; exceeding the FS_SPD value will switch the device into full-

step mode. The SpdCalc() function is provided to convert steps/s values into appropriate integer values for this function.
_Run(byte, unsigned long);

STEP_CLOCK puts the device in external step clocking mode. When active, pin 13, STCK, becomes the step clock for the device, and steps it in the direction (set by the FWD and REV constants) imposed by the call of this function. Motion commands (RUN, MOVE, etc) will cause the device to exit step clocking mode.
_Step_Clock(byte);

MOVE will send the motor n_step steps (size based on step mode) in the direction imposed by dir (FWD or REV constants may be used). The motor will accelerate according the acceleration and deceleration curves, and will run at MAX_SPEED. Stepping mode will adhere to FS_SPD value, as well.
_Move(byte, unsigned long);

GOTO operates much like MOVE, except it produces absolute motion instead of relative motion. The motor will be moved to the indicated position in the shortest possible fashion.
_GoTo(unsigned long);

Same as GOTO, but with user constrained rotational direction.
_GoTo_DIR(byte, unsigned long);

GoUntil will set the motor running with direction dir (REV or FWD) until a falling edge is detected on the SW pin. Depending on bit SW_MODE in CONFIG, either a hard stop or a soft stop is performed at the falling edge, and depending on the value of act (either RESET or COPY) the value in the ABS_POS register is either RESET to 0 or COPY-ed into the MARK register.
_GoUntil(byte, byte, unsigned long);

Similar in nature to GoUntil, ReleaseSW produces motion at the higher of two speeds: the value in MIN_SPEED or 5 steps/s. The motor continues to run at this speed until a rising edge is detected on the switch input, then a hard stop is performed and the ABS_POS register is either COPY-ed into MARK or RESET to 0, depending on whether RESET or COPY was passed to the function for act.
_ReleaseSW(byte, byte);

GoHome is equivalent to GoTo(0), but requires less time to send. Note that no direction is provided; motion occurs through shortest path. If a direction is required, use GoTo_DIR().
_GoHome();

GoMark is equivalent to GoTo(MARK), but requires less time to send. Note that no direction is provided; motion occurs through shortest path. If a direction is required, use GoTo_DIR().
_GoMark();

Sets the ABS_POS register to 0, effectively declaring the current position to be "HOME".
_ResetPos();

Reset device to power up conditions. Equivalent to toggling the STBY pin or cycling power.
_ResetDev();

Bring the motor to a halt using the deceleration curve.
_SoftStop();

Stop the motor with infinite deceleration.
_HardStop();

Decelerate the motor and put the bridges in Hi-Z state.
_SoftHiZ();

Put the bridges in Hi-Z state immediately with no deceleration.
_HardHiZ();