Finger Navigation Datasheet LPNAV V 1.10

Copyright © 2011-2012 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	API Memory (Bytes)		Pins
	Flash	RAM	
CYONSFN3050	2204	45	-

Features and Overview

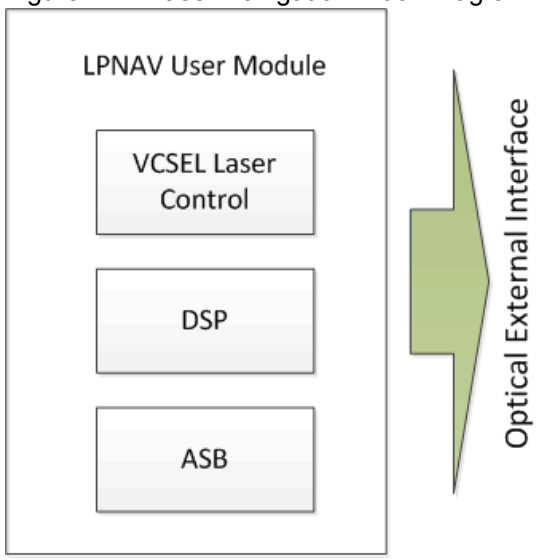
- Control of laser navigation engine and on-chip power management
- Resolution (counts per inch) control
- Continuous variable resolution setting from 1 to 3200 dpi
- 360 degree navigation support
- Independent X and Y resolution control
- Flexible track modes
- Automatic laser control for eye-safe operation
- Power consumption optimized for mobile device applications

The LPNAV User Module enables designers to control and configure the settings for a laser navigation system.

Functional Description

The purpose of the LPNAV User Module is to give access to the navigation and laser control features and functions of a device.

Figure 1. Laser Navigation Block Diagram



The LPNAV User Module uses the laser navigation block that consists of Vertical Cavity Surface-Emitting Laser (VCSEL) driver, Digital Signal Processor (DSP), and Analog Super Block (ASB). Interface to the navigation block is through registers accessible to the PSoC Block.

Tracking Control functionality performs initialization of tracking registers; tracks all stabilization process completion; trims the power system, operating and eye safe VCSEL current for each mode; and sets resolution scaling.

Laser Control functionality trims Automatic Gain Control (AGC), ASB and VCSEL for laser work, enables you through API to configure laser eye safe and laser operating current for each tracking mode, and turns on/off AGC control.

The LPNAV User Module tracks your finger motion and reports delta X and delta Y displacement data, which enables 360^{o} navigation of a cursor on a system display.

The LPNAV User Module is designed to track finger motion by sensing a laser speckle scattered from a human finger which is illuminated by a VCSEL. VCSEL driver circuitry turns on an external VCSEL die; the VCSEL illuminates a moving finger. The laser speckle scattered back from the finger is received by the Photodiode Array (PDA) on the device. The ASB block converts the photo-current from the PDA to voltage signals, which are further converted into digital signals through an analog-to-digital converter (ADC). The DSP block processes the digital signals into finger motion data. The processed motion data can be passed to an external host using I2C User Module, SPI User Module, or by manually toggling GPIOs.

Operation Modes

The device has two operating modes. These are:

- Active Mode

  Upon applying power to the device, the supervisory ROM (SROM) initializes the hardware and then runs a user-generated application code from FLASH memory. This mode covers all conditions where the sensor detects motion. This mode can be further subdivided into several tracking modes that are optimized to minimize the average power consumption at different tracking speeds. This mode is entered by applying power to the device with a user-generated application code that is pre-programmed to its FLASH memory.

- Hibernate Mode

  This mode provides the lowest power mode of operation. It uses the "I2C sleep" mode and powers down the remaining sensor blocks. It does not detect motion. This mode is entered by writing to a set of power control registers to power down the sensor blocks, and then writing to another set of registers to put the PSoC portion of the chip in its "I2C sleep" mode. This mode is exited by waking up the PSoC portion from its I2C sleep mode through an interrupt, then writing to a set of power control registers to power up the sensor blocks in proper sequence. Refer to the OvationONS II Laser Navigation System-on-Chip Technical Reference Manualfor further information about "I2C sleep" mode.

Placement

The LPNAV User Module can be placed in the dedicated block of CYONSFN3050 only.

Parameters and Resources

There are no configurable user module parameters.

Application Programming Interface

The Application Programming Interface (API) functions are provided as part of the user module to allow you to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the include files.

Only one instance of this user module can be placed in the project and this also applies to loadable configurations.

Note

****** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must also ensure their code observes the policy. Though some user module API function may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

LPNAV API

Function	Description
General Purpose API	
void **LPNAV_Start**(void)	Starts the power system.
void **LPNAV_Stop**(void)	Empty API for consistency with other user modules.
BYTE **LPNAV_bReadRegister**(BYTE bioBank, BYTE bAddr)	Returns the value of the register specified by the arguments.
void **LPNAV_bWriteRegister**(BYTE bioBank, BYTE bAddr, BYTE bData)	Writes a value to the register specified in the arguments. No error checking or write protection is used.
BOOL **LPNAV_fCheckInitDone**(void)	Returns flag indicating if power system initialization is done.
Tracking API	
BYTE **LPNAV_bTrackInit** (void)	Initializes the tracking engine with the built-in register settings.
void **LPNAV_TrackStart**(void)	Starts the navigation engine.
void **LPNAV_TrackStop**(void)	Stops the navigation engine.
void **LPNAV_SetResolution**(WORD wDPI)	Sets desired optical sensor resolution for both X and Y axes.
void **LPNAV_SetXResolution**(WORD wXDPI)	Sets desired optical sensor resolution for both X-axis.
void **LPNAV_SetYResolution**(WORD wYDPI)	Sets desired optical sensor resolution for both Y-axis.
WORD **LPNAV_wReadResolution**(void)	Returns current resolution. In the axes resolutions are different it returns X-axis resolution.

Function	Description
WORD **LPNAV_wReadXResolution**(void)	Returns current X-axis resolution.
WORD **LPNAV_wReadYResolution**(void)	Returns current Y-axis resolution.
BOOL **LPNAV_fReadXYCounts**(POSITION* pSXYData)	Reads XY counts and updates the pSXYData structure. Returns false if both X and Y are zero, and true otherwise.
void **LPNAV_TrackTransitionEnableInt**(void)	Enables Global subinterrupt: Track Transition.
void **LPNAV_TrackTransitionDisableInt**(void)	Disables Global subinterrupt: Track Transition.
void **LPNAV_ForceTrackMode**(BYTE bTrackMode)	Go to the track mode specified by the given index.
Laser Control API	
void **LPNAV_LaserStart**(void)	Turns on the laser driver.
void **LPNAV_LaserStop**(void)	Turns off the laser driver.
void **LPNAV_Startup**(void)	Powers up sensor, initially or after a successful power down.
void **LPNAV_ShutDown**(void)	Triggers a shutdown of the power system.
void **LPNAV_VcselErrorEnableInt**(void)	Enables the general interrupt that is generated by the navigation engine as a result of a VCSEL error.
void **LPNAV_VcselErrorDisableInt**(void)	Disables the general interrupt that is generated by the navigation engine as a result of a VCSEL error.
void **LPNAV_LaserSetPower**(BYTE bPowerSetting)	Manually change the laser power.
void **LPNAV_LaserAgcControl**(BOOL fAgc)	Turns on/off the AGC loop.
void **LPNAV_LaserModulationControl**(BOOL fOnOff)	Turns on/off the laser modulation.
BYTE **LPNAV_bLaserReadPower**(void)	Returns the current laser power setting.
BYTE **LPNAV_bLaserReadEyesafePower**(void)	Returns the laser driver eye-safe level for this sensor part.
BYTE **LPNAV_bLaserReadMaximumPower**(void)	Returns the laser driver maximum current setting for this sensor part.
void **LPNAV_LaserSetTestMode**(void)	Sets the laser to CW mode, without modulation and without AGC.
void **LPNAV_GlobalEnableInt**(void)	Enables Global interrupt.
void **LPNAV_GlobalDisableInt**(void)	Disables Global interrupt.
BOOL **LPNAV_fHadVcselError**(void)	Checks VCSEL Error status bit.
BOOL **LPNAV_fHadTrackingBlankout**(void)	Returns flag indicating if tracking had blankout due to weak signal.

Function	Description
void **LPNAV_ResetVcsel**(void)	Performs DSP soft reset.
BYTE **LPNAV_bfReadCurrentPreviousState**(void)	Reads previous and current track state in sensor.
void **LPNAV_LaserSetEyesafePower**(BYTE bPowerSetting)	Sets eye-safety VCSEL current value.
void **LPNAV_LaserSetMaximumPower**(BYTE bPowerSetting)	Assigns operating VCSEL current settings for all track modes.

General Purpose API

LPNAV_Start

Description:

Configures DSP clock based on selected IMO setting.

C Prototype:

```
void  LPNAV_Start(void)
```

Assembly:

```
lcall  LPNAV_Start
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_Stop

Description:

Empty API for consistency with other user modules.

C Prototype:

```
void  LPNAV_Stop(void)
```

Assembly:

```
lcall  LPNAV_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_bReadRegister

Description:

Returns the value of the register specified by the arguments.

C Prototype:

```
BYTE LPNAV_bReadRegister(BYTE bioBank, BYTE bAddr)
```

Assembly:

```
mov   X, bAddr
mov   A, bioBank
lcall LPNAV_bReadRegister
```

Parameters:

bioBank is the M8C register bank, bAddr is the address in that bank.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_bWriteRegister

Description:

Writes a value to the register specified in the arguments. No error checking or write protection is used.

C Prototype:

```
void LPNAV_bWriteRegister(BYTE bioBank, BYTE bAddr, BYTE bData)
```

Assembly:

```
mov  A, bData
push A
mov  A, bAddr
push A
mov  A, bAddr
push A
lcall LPNAV_bWriteRegister
```

Parameters:

bioBank is the M8C register bank, bAddr is the address in that bank, and bData is the value to be written.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_fCheckInitDone

Description:

Returns flag to indicate the status of power system initialization.

C Prototype:

```
BOOL LPNAV_fCheckInitDone(void)
```

Assembly:

```
lcall LPNAV_fCheckInitDone
```

Parameter:

None

Return Value:

Symbolic Name	Value	Description
LPNAV_INIT_NOT_DONE	0	Initialization not done
LPNAV_INIT_IS_DONE	1	Initialization is done

Side Effects:

See Note ** at the beginning of the API section.

Tracking API

LPNAV_bTrackInit

Description:

Initializes the tracking engine with the built-in register settings. These settings (flash table) is set by default when the user module is placed and code is generated. This guarantees proper operation after the boot sequence is completed.

C Prototype:

```
BYTE  LPNAV_bTrackInit(void)
```

Assembly:

```
lcall  LPNAV_bTrackInit
;now error code is passed through A register
```

Parameters:

None

Return Value:

Symbolic Name	Value	Description
LPNAV_NO_ERROR	0	No Error
LPNAV_VCSEL_CHECKSUM_ERROR	1	VCSEL checksum error

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_TrackStart

Description:

Starts the navigation engine. The LPNAV_bTrackInit() should be called before calling this function. This enables you to modify the tracking registers according to your required values before you call LPNAV_TrackStart().

C Prototype:

```
void  LPNAV_TrackStart(void)
```

Assembly:

```
lcall  LPNAV_TrackStart
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_TrackStop

Description:

Stops the navigation engine. After the NAV engine is stopped, to restart the engine, call LPNAV_bTrackInit(). Now you have the option to modify tracking registers to your required values, before calling LPNAV_TrackStart().

C Prototype:

```
void LPNAV_TrackStop(void)
```

Assembly:

```
lcall LPNAV_TrackStop
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_SetResolution

Description:

Converts the new resolution in DPI to the appropriate X-axis and Y-axis resolution scaling register values (the same value for both registers).

C Prototype:

```
void LPNAV_SetResolution(WORD wDPI)
```

Assembly:

```
mov   X, >wDPI
mov   A, <wDPI
lcall LPNAV_SetResolution
```

Parameters:

wDPI - specifies the resolution in DPI for both X and Y directions (X <= MSB; A <= LSB)

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_SetXResolution

Description:

Converts the new x-axis resolution in DPI to the appropriate X-axis resolution scaling register value.

C Prototype:

```
void LPNAV_SetXResolution(WORD wXDPI)
```

Assembly:

```
mov   X, >wXDPI
mov   A, <wXDPI
lcall LPNAV_SetXResolution
```

Parameter:

wXDPI - specifies the X-axis resolution in DPI (X <= MSB; A <= LSB)

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_SetYResolution

Description:

Converts the new y-axis resolution in DPI to the appropriate Y-axis resolution scaling register value.

C Prototype:

```
void LPNAV_SetYResolution(WORD wYDPI)
```

Assembly:

```
mov   X, >wYDPI
mov   A, <wYDPI
lcall LPNAV_SetYResolution
```

Parameters:

wYDPI - specify the Y-axis resolution in DPI (X <= MSB; A <= LSB)

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_wReadResolution

Description:

This function converts the value in the resolution register to the equivalent DPI value and returns the X- and Y-axis DPI value. This function is identical to LPNAV_ReadXResolution. If you set the X and Y resolution with the LPNAV_SetResolution() API function, the X and Y axes are both set to the same resolution and this function returns the resolution. If the values are not the same, this function returns the X-axis resolution.

C Prototype:

```
WORD LPNAV_wReadResolution(void)
```

Assembly:

```
lcall LPNAV_wReadResolution
; now X contains MSB and A - LSB of returned value
```

Parameters:

None

Return Value:

Returns the current X-axis and Y-axis resolution setting in DPI. X <= MSB part of result; A <= LSB part of result

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_wReadXResolution

Description:

Returns current X-axis resolution setting in DPI

C Prototype:

```
WORD LPNAV_wReadXResolution(void)
```

Assembly:

```
lcall LPNAV_wReadXResolution
; now X contains MSB and A - LSB of returned value
```

Parameters:

None

Return Value:

Returns the current X-axis resolution setting in DPI. X <= MSB part of result; A <= LSB part of result.

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_wReadYResolution

Description:

Returns current Y-axis resolution setting in DPI

C Prototype:

```
WORD LPNAV_wReadYResolution(void)
```

Assembly:

```
lcall LPNAV_wReadYResolution
; now X contains MSB and A - LSB of returned value
```

Parameters:

None

Return Value:

Returns the current y-axis resolution setting in DPI.

X <= MSB part of result; A <= LSB part of result

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_fReadXYCounts

Description:

Reads XY counts and updates the SXYData structure.

C Prototype:

```
BOOL LPNAV_fReadXYCounts(POSITION* pSXYData)
```

Assembly:

```
mov X, <pSXYData
mov A, >pSXYData
lcall LPNAV_fReadXYCounts
; now X contains MSB and A - LSB of returned value
```

Parameters:

POSITION* pSXYData. SXYData points to X and Y movement counts from the navigation sensor.

Return Value:

Returns false if both X and Y are zero, and true otherwise.

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_ForceTrackMode

Description:

Goes to the track mode specified by the given index. This function should be used after the LPNAV_TrackStart function is called. Tracking mode with higher index uses higher sampling rate.

C Prototype:

```
void LPNAV_ForceTrackMode(BYTE bTrackMode)
```

Assembly:

```
mov  A, bTrackMode
lcall LPNAV_ForceTrackMode
```

Parameter:

bTrackMode - index of the tracking mode (0 – 3).

Symbolic Name	Value	Description
LPNAV_TRACK_MODE_0	0	Track mode 0
LPNAV_TRACK_MODE_1	1	Track mode 1
LPNAV_TRACK_MODE_2	2	Track mode 2
LPNAV_TRACK_MODE_3	3	Track mode 3

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

Laser Control API

LPNAV_LaserStart

Description:

Turns on the laser driver. If AGC is off, or if CW mode is on, then the initial power resulting from this command is zero. The firmware must then set the laser power using the LPNAV_LaserSetPower() call.

C Prototype:

```
void  LPNAV_LaserStart(void)
```

Assembly:

```
lcall  LPNAV_LaserStart
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_LaserStop

Description:

Turns off the laser driver.

C Prototype:

```
void  LPNAV_LaserStop(void)
```

Assembly:

```
lcall  LPNAV_LaserStop
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_Startup

Description:

Powers up sensor, initially or after a successful power down. Complementary with LPNAV_Shutdown(). This function is used to bring the sensor in and out of hibernate mode.

C Prototype:

```
void  LPNAV_Startup(void)
```

Assembly:

```
lcall  LPNAV_Startup
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_ShutDown

Description:

Triggers a shutdown of the power system. Complementary with LPNAV_Startup(). This function is used to bring the sensor in and out of hibernate mode.

C Prototype:

```
void  LPNAV_ShutDown(void)
```

Assembly:

```
lcall  LPNAV_ShutDown
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_VcselErrorEnableInt

Description:

Enables the general interrupt that is generated by the navigation engine as a result of a VCSEL error.

Note that the general interrupt can also be generated by the power system. It is the firmware's responsibility to manage the different scenarios that are covered by this interrupt.

C Prototype:

```
void LPNAV_VcselErrorEnableInt(void)
```

Assembly:

```
lcall LPNAV_VcselErrorEnableInt
```

Parameter:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_VcselErrorDisableInt

Description:

Disables the VCSEL error interrupt that is generated by the Navigation module.

C Prototype:

```
void LPNAV_VcselErrorDisableInt(void)
```

Assembly:

```
lcall LPNAV_VcselErrorDisableInt
```

Parameter:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_LaserSetPower

Description:

Turns on the laser driver. If AGC is off, or if CW mode is on, then the initial power resulting from this command is zero. The firmware must then set the laser power using the LPNAV_LaserSetPower() call.

C Prototype:

```
void  LPNAV_LaserSetPower(BYTE bPowerSetting)
```

Assembly:

```
mov A, bPowerSetting
lcall  LPNAV_LaserSetPower
```

Parameters:

bPowerSetting - laser driver current setting. Range is 0 – 255.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_LaserAgcControl

Description:

Turns on/off the AGC loop.

C Prototype:

```
void  LPNAV_LaserAGCControl(BOOL fAgc)
```

Assembly:

```
mov   A, fAgc
lcall  LPNAV_LaserAGCControl
```

Parameters:

fAGC - AGC loop Settings:

Symbolic Name	Value	Description
LPNAV_AGC_LOOP_OFF	0	AGC loop off
LPNAV_AGC_LOOP_ON	1	AGC loop on

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_LaserModulationControl

Description:

Turns on or off the laser modulation.

C Prototype:

```
void  LPNAV_LaserModulationControl(BOOL fOnOff)
```

Assembly:

```
mov   A, fOnOff
lcall  LPNAV_LaserModulationControl
```

Parameters:

fOnOff – Modulation Control Settings: if non-zero value then it takes pulsed mode, otherwise continuous-wave (CW) mode.

Symbolic Name	Value	Description
LPNAV_PULSED_MODE	0	Pulsed mode
LPNAV_CW_MODE	1	CW mode

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_bLaserReadPower

Description:

Returns the current laser power setting.

C Prototype:

```
BYTE  LPNAV_bLaserReadPower(void)
```

Assembly:

```
lcall  LPNAV_bLaserReadPower
```

```
; now returned value is in A register
```

Parameters:

None

Return Value:

A <= current laser driver setting

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_bLaserReadEyesafePower

Description:

Returns the laser driver eye-safe level for this sensor part.

C Prototype:

```
BYTE  LPNAV_bLaserReadEyesafePower(void)
```

Assembly:

```
lcall  LPNAV_bLaserReadEyesafePower
; now returned value is in A register
```

Parameters:

None

Return Value:

A <= laser driver eye-safe level for this sensor part.

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_bLaserReadMaximumPower

Description:

Returns the laser driver maximum current setting for this sensor part.

C Prototype:

```
BYTE  LPNAV_bLaserReadMaximumPower(void)
```

Assembly:

```
lcall  LPNAV_bLaserReadMaximumPower
; now returned value is in A register
```

Parameters:

None

Return Value:

A <= laser driver maximum current setting for this sensor part

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_LaserSetTestMode

Description:

This command sets the laser to CW mode, without modulation and without AGC. These settings are required to test the laser output power for eye safety. To exit this mode, call LPNAV_LaserStart.

C Prototype:

```
void  LPNAV_LaserSetTestMode(void)
```

Assembly:

```
lcall  LPNAV_LaserSetTestMode
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_GlobalEnableInt

Description:

Enables Global interrupt.

C Prototype:

```
void  LPNAV_GlobalEnableInt(void)
```

Assembly:

```
lcall  LPNAV_GlobalEnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_GlobalDisableInt

Description:

Disables Global interrupt.

C Prototype:

```
void  LPNAV_GlobalDisableInt(void)
```

Assembly:

```
lcall  LPNAV_GlobalDisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_fHadVcselError

Description:

Checks the VCSEL Error status bit.

C Prototype:

```
BOOL  LPNAV_fHadVcselError(void)
```

Assembly:

```
lcall  LPNAV_fHadVcselError
```

Parameters:

None

Return Value:

Returns false if there is no VCSEL Error, and true otherwise.

Symbolic Name	Value	Description
LPNAV_VCSEL_NO_ERROR	0	VCSEL No Error
LPNAV_VCSEL_ERROR	1	VCSEL Error

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_fHadTrackingBlankout

Description:

Returns flag a to indicate if tracking had blankout due to weak signal.

C Prototype:

```
BOOL  LPNAV_fHadTrackingBlankout(void)
```

Assembly:

```
lcall  LPNAV_fHadTrackingBlankout
```

Parameters:

None

Return Value:

Returns false if there is no Track Blackout, and true otherwise.

Symbolic Name	Value	Description
LPNAV_NO_TRACK_BLACKOUT	0	No Track Blackout
LPNAV_TRACK_BLACKOUT	1	Track Blackout

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_ResetVcselt

Description:

Performs DSP soft reset.

C Prototype:

```
void  LPNAV_ResetVcsel(void)
```

Assembly:

```
lcall  LPNAV_ResetVcsel
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_bfReadCurrentPreviousState

Description:

Performs DSP soft reset.

C Prototype:

```
BYTE  LPNAV_bfReadCurrentPreviousState(void)
```

Assembly:

```
lcall  LPNAV_bfReadCurrentPreviousState
```

Parameters:

None

Return Value:

A <= first nibble - CurState, second nibble - PrevState.

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_LaserSetEyesafePower

Description:

Sets eye-safety VCSEL current value.

C Prototype:

```
void  LPNAV_LaserSetEyesafePower(BYTE bPowerSetting)
```

Assembly:

```
mov A, bPowerSetting
lcall  LPNAV_LaserSetEyesafePower
```

Parameters:

bPowerSetting - eye-safety VCSEL current settings. Range is 0 – 255.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

LPNAV_LaserSetMaximumPower

Description:

Assigns operating VCSEL current settings for all track modes.

C Prototype:

```
void  LPNAV_LaserSetMaximumPower(BYTE bPowerSetting)
```

Assembly:

```
mov A, bPowerSetting
lcall  LPNAV_LaserSetMaximumPower
```

Parameters:

bPowerSetting - Maximum setting in unscaled counts. Range is 0 – 255.

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

Sample Firmware Source Code

The C code given here shows you how to use the LPNAV User Module:

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

#define ENDLESS_LOOP 1
#define WIRED 0

void main(void)
{

//
// Integers to hold our X an Y counts read from the sensor.
//

int iX, iY;

//
// POSITION is defined in LPNAV.h.
//
// typedef struct {
// INT x;
// INT y;
// } POSITION;
//

POSITION XYData;

LPNAV_Start();
//
// Start the OvationONS II DSP Tracking Mode.
//
// Always call these first three NAV user modules in order.
//
// 1. LPNAV_bTrackInit()
// 2. LPNAV_LaserStart()
// 3. LPNAV_TrackStart()
//

LPNAV_bTrackInit();
LPNAV_LaserStart();
LPNAV_TrackStart();

do
{
//
// Read the change in X and Y counts from the last read.
//
LPNAV_fReadXYCounts(&XYData);
//
// pXYData now points to X and Y movment counts from the navigation sensor.
// Send XYData.x and XYData.y to USB, SPI, or other reporting protocol.
//
```

```
iX = XYData.x;
iY = XYData.y;

} while (ENDLESS_LOOP);

}
```

The Assembly code given here implements a function similar to the C example:

```
;-----------------------------------------------------------------------
; Assembly main line
;-----------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

area bss
export _LPNAVInfo
export LPNAVInfo
_LPNAVInfo:
LPNAVInfo: blk 4

area text
export LPNAV_demo_init
export _LPNAV_demo_init
export LPNAV_demo
export _LPNAV_demo

export _main

_main:

    ; Insert your main assembly code here.
    lcall LPNAV_demo_init
.loop:
    lcall LPNAV_demo
jmp .loop
.terminate:
    jmp .terminate

LPNAV_demo_init:
_LPNAV_demo_init:

lcall LPNAV_Start
lcall LPNAV_bTrackInit
lcall LPNAV_LaserStart
lcall LPNAV_TrackStart

ret

LPNAV_demo:
_LPNAV_demo:
mov X, <LPNAVInfo
mov A, >LPNAVInfo
```

```
lcall LPNAV_fReadXYCounts
; data will be located in LPNAVInfo location
ret
```

Configuration Registers

The following registers are configured in this user module. Symbolic names for these registers are defined in the user module instance C and assembly language interface files (the *.h* and *.inc* files).

Table 1. INTR_MASK_REG, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	Reserved				st_forc			vcsel_err

st_forc - This interrupt occurs each time there is a change in the track mode.

vcsel_err - Whenever this interrupt is set, it means that VCSEL is ON for more than the required time interval. This signal makes a transition from 0 to 1 to indicate error.

Table 2. X_CNT_REG_BUF0, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	x_cnt_reg_buf (LSB)							

x_cnt_reg_buf - X count buffer register (LSB).

Table 3. X_CNT_REG_BUF1, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	x_cnt_reg_buf (MSB)							

x_cnt_reg_buf - X count buffer register (MSB).

Table 4. Y_CNT_REG_BUF0, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	y_cnt_reg_buf (LSB)							

y_cnt_reg_buf - X count buffer register (LSB).

Table 5. Y_CNT_REG_BUF1, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	y_cnt_reg_buf (MSB)							

y_cnt_reg_buf - X count buffer register (MSB).

Table 6. RES_SCAL_DX0, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	res_scal_dx (LSB)							

res_scal_dx - used to convert Counts to DX. These can be updated whenever needed. These registers can be updated at any time, but their visibility in the algorithm is synchronized to the next block boundary.

Table 7. RES_SCAL_DX1, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	res_scal_dx (MSB)							

res_scal_dx - used to convert Counts to DX. These can be updated as needed. These registers can be updated at any time, but their visibility in the algorithm is synchronized to the next block boundary.

Table 8. RES_SCAL_DY0, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	res_scal_dy (LSB)							

res_scal_dy - used to convert Counts to DY. These can be updated as needed. These registers can be updated at any time, but their visibility in the algorithm is synchronized to the next block boundary.

Table 9. RES_SCAL_DY1, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	res_scal_dy (MSB)							

res_scal_dy - used to convert Counts to DY. These can be updated as needed. These registers can be updated at any time, but their visibility in the algorithm is synchronized to the next block boundary.

Table 10. PWR_DSP_CTRL_REG, Bank 1

Bit	7	6	5	4	3	2	1	0
Value		res_cpi_limit_reg						

res_cpi_limit_reg - resolution/CPI limiter data.

Table 11. THRESHOLD1, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	threshold1							

threshold1 - the higher threshold used in lift detection logic.

Table 12. THRESHOLD2, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	threshold2							

threshold2 - the lower threshold used in lift detection logic.

Table 13. INT_MSK3, Bank 0

Bit	7	6	5	4	3	2	1	0
Value	Reserved						glb_ons	wkp_ons

glb_ons - mask optical navigation system global interrupt.

wkp_ons - mask optical navigation system wakeup interrupt.

Table 14. EYE_CW_VCSEL_CUR0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	eye_cw_vcsel_cur							

eye_cw_vcsel_cur0 - continuous wave eye-safety VCSEL.

Table 15. VCSEL_DAC_CURRENT_AGC, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	vcsel_dac_current_agc							

vcsel_dac_current_agc - This is the VCSEL Current given to the VCSEL power controller block. This is updated at every block boundary. This is writable by both CPU and the internal logic. The CPU is given the highest priority.

Table 16. GLOBAL_CONFIG_REG0, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	Reserved							dsp_start

dsp_start - starts tracking engine.

Table 17. BG_BUFF_LVD_TRIM, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	Reserved					LVD_offset_lv_trim		

LVD_offset_lv_trim - LVD trimming setting.

Table 18. TMx_EYE_SAFE_CURR0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	tmx_eyesaf_curr							

tmx_eyesaf_curr - Eye-safety current for corresponding track mode.

Table 19. TMx_MAX_VCSEL_PWR0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	tmx_max_vcsel_pwr							

tmx_max_vcsel_pwr - max integration time. This is selected based on the track mode.

Table 20. MAX_CW_VCSEL_CUR0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	max_cw_vcsel_cur							

max_cw_vcsel_cur0 - continuous wave Max VCSEL.

Table 21. AVERAGED_AGC_TAR_MIN0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	averaged_agc_tar_min (LSB)							

averaged_agc_tar_min - averaged AGC target min for the computation of Gain for AGC and differential channels; integration time, and VCSEL current.

Table 22. AVERAGED_AGC_TAR_MIN1, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	Reserved							averaged_agc_tar_min (MSB)

averaged_agc_tar_min - averaged AGC target minimum for the computation of Gain for AGC and differential channels; integration time, and VCSEL current.

Table 23. AVERAGED_AGC_TAR_MAX0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	averaged_agc_tar_max (LSB)							

averaged_agc_tar_max - averaged AGC target maximum for the computation of Gain for AGC and differential channels; integration time, and VCSEL current.

Table 24. PEER_PRESSURE_PIPELINE_DEPTH, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	Reserved				peer_pressure_pipeline_depth			

peer_pressure_pipeline_depth - peer pressure depth to be used. If this is programmed to 0, peer pressure is disabled.

Table 25. TMx_BLNK_HYST, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	tm**x**_blnk_hyst							

tm**x**_blnk_hyst - blanking hysteresis value for each tracking mode.

Table 26. TMx_BLNK_THRES, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	tm**x**_blnk_thres							

tm**x**_blnk_thres - blanking threshold value for each tracking mode (blank out signal if dpp < Blank Thrsh).

Table 27. ADC_CTRL_REG, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	Reserved	Reserved	vbias_en	winshift_hw			winalt_hv	

vbias_en - When high, enables the path for vbias to ADC. When low, enables the path for vbias to CDS test input.

winshift_hv - This register is for shifting the full ADC conversion window.

winalt_hv - This register is for altering the range of the ADC conversion window.

Table 28. CURRENT_VCSEL_OPPT_REG0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	current_vcsel_oppt_reg							

current_vcsel_oppt_reg - Current VCSEL operating current value. These registers are generated based on the VCSEL_ON signal from the analog block.

Table 29. CURRENT_VCSEL_EYESAF_REG0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	current_vcsel_eyesaf_reg							

current_vcsel_eyesaf_reg - current VCSEL eye-safety current value. These registers are generated based on the VCSEL_ON signal from the analog block.

Version History

Version	Originator	Description
1.00	DHA	Initial Version
1.10	DHA	1. mulu_16x16_32 function was replaced by mul_16x16_32. 2. Native paging is restored before mul_16x16_32 call. 3. Some user module variables moved from InterruptRAM to ramTmp and ramCalc RAMs. 4. Explained limitations of dynamic reconfiguration in datasheet.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2011-2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.