Preliminary

Laser Navigation Datasheet LaserNAV V 1.10

Copyright © 2009-2012 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	API Memory (Bytes)		Pins
	Flash	RAM	
CYONS2000, CYONS2001, CYONS2010, CYONS2011, CYONS2100, CYONS2101, CYONS2110, CYONSFN2051, CYONSFN2053, CYONSFN2061, CYONSFN2151, CYONSFN2161, CYONSFN2162, CYONSTB2010, CYONSTB2011, CYONSFN2010-BFXC, CYONSCN2024-BFXC, CYONSCN2028-BFXC, CYONSCN2020-BFXC, CYONSKN2033-BFXC, CYONSKN2035-BFXC, CYONSKN2030-BFXC, CYONSTN2040	1207	17	-

Features and Overview

- Control of laser navigation engine and on-chip power management
- Resolution (counts per inch) control
- Flexible track and sleep modes
- Automatic laser control for eye-safe operation
- USB 5V, battery, or 3V external power operation, when supported by device.

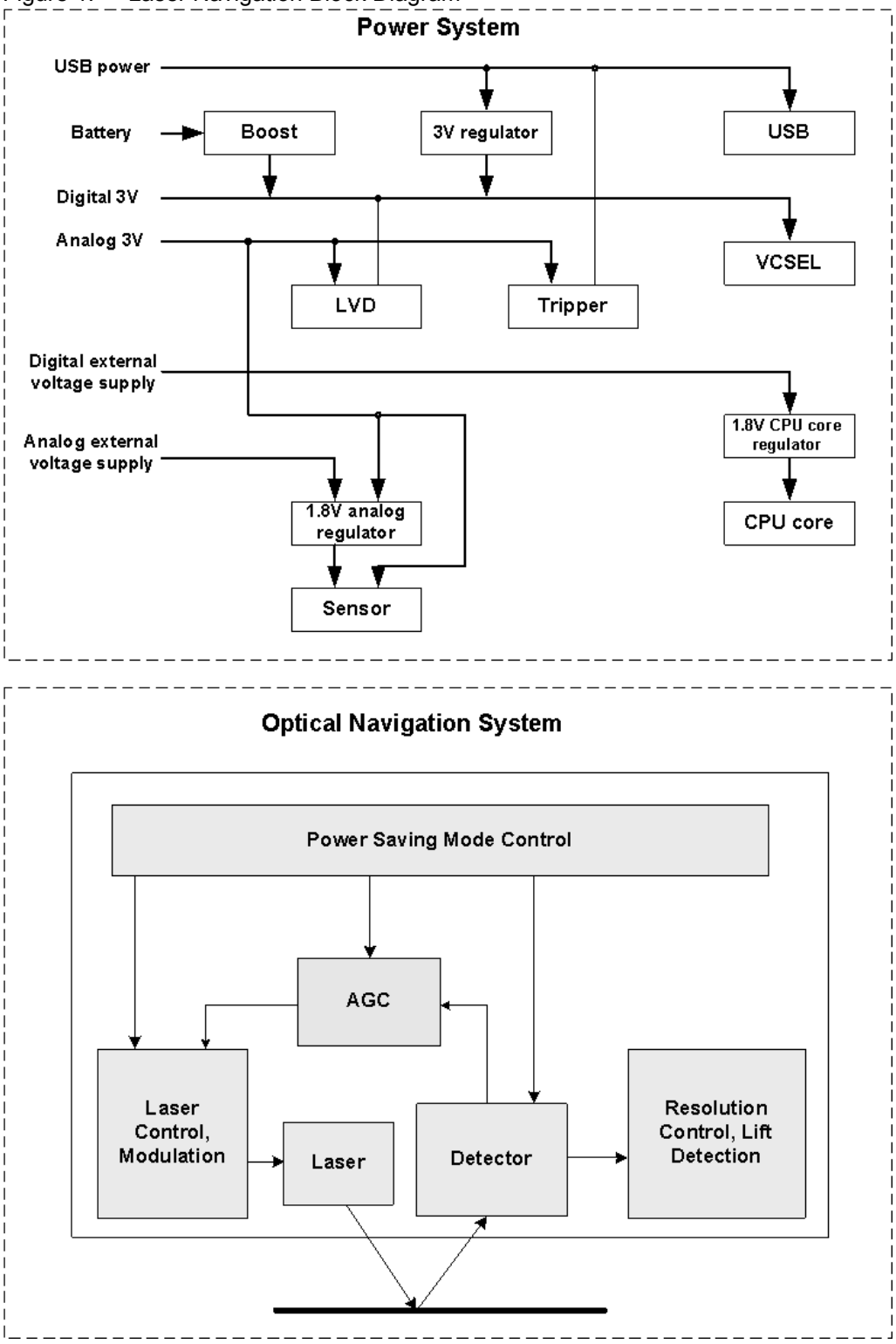
The LaserNAV User Module enables you to control and configure the settings for the laser navigation and power system. Finger navigation and wired/wireless/hybrid mouse and trackball solutions are all supported.

The optical navigation system can be divided into three blocks:

- The TRACKING SYSTEM CONTROL block manages surface tracking, including the resolution in the x and y directions.
- The POWER-SAVING MODE CONTROL block configures the various sleep and tracking modes available to the sensor. This block can force the sensor to a certain tracking or sleep mode, and also can set the parameters used by the sensor as it automatically transitions between active and sleep states.
- The LASER CONTROL block controls the settings of the laser, allowing the user to enable/disable AGC and laser modulation, and to enter laser test mode. Laser eye safe requirements are available as a semiautomatic feature of the laser control APIs. Laser calibration is stored in protected rows of flash and can be obtained on the fly, as an API function.

Additionally the Automatic Gain Control (AGC) block uses velocity data from the DSP block to control the sensor sampling rate. At low speeds the sensor can sample the input signals at a low rate, and hence lower speed, without sacrificing accuracy. At higher speeds the sampling rate must increase to keep up with motion. Since the sampling rate is intimately tied to the frame integration time and the DSP calculations are independent of sampling rate, it makes sense to have the AGC block control the sampling rate.

Figure 1. Laser Navigation Block Diagram



Functional Description

The purpose of the Power User Module is to give access to the unique navigation, power supply features and functions of the device system. The device is capable of operating from one or two batteries or from a USB interface. It can also run off of a fixed 3.3 volt supply.

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

The resolution limits for Ovation devices are listed in the following table:

Table 1. Resolution Limits for Ovation Devices

Device	Supported Resolution
CYONS2000-LBXC	400, 800, 1600 dpi
CYONS2001-LBXC	400, 800, 1600dpi
CYONS2100-LBXC	400-3200dpi
CYONS2101-LBXC	400-3200dpi
CYONS2110-LBXC	400-3200dpi
CYONS2010-LBXC	1-1600
CYONS2011-LBXC	1-1600
CYONSFN2051-LBXC	1-3200
CYONSFN2061-LBXC	1-1600
CYONSFN2151-LBXC	1-3200
CYONSFN2161-LBXC	1-3200
CYONSFN2162-LBXC	1-3200
CYONSFN2053-LBXC	1-1600
CYONSTB2010-LBXC	1-1600
CYONSTB2011-LBXC	1-1600

Power Saving functionality provides system with facilities to configure each of sleep mode, set sleep delay, enable/disable sleep, wake interrupts, check tripper and LVD outputs.

Laser Control functionality trims AGC, ASB and VCSEL for laser work, gives user an opportunity via API to configure laser eye safe and laser operating current for each tracking mode, turns on/off AGC control.

The following four flowcharts present various reset scenarios:

Figure 2. Figure 2. Starting the Sensor from Power-On-Reset

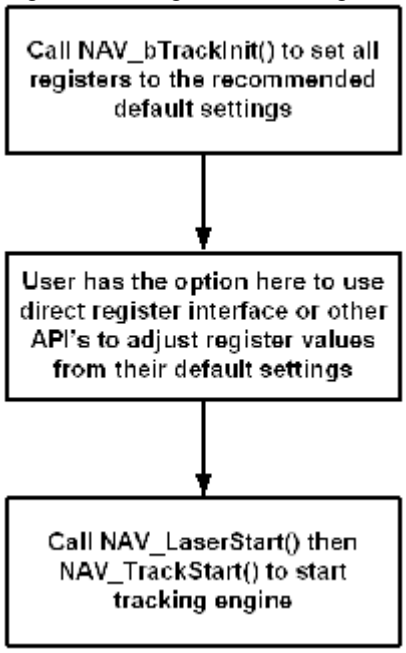


Figure 3. Figure 3. Restarting or Resetting the Sensor Without Cycling Power to the Part

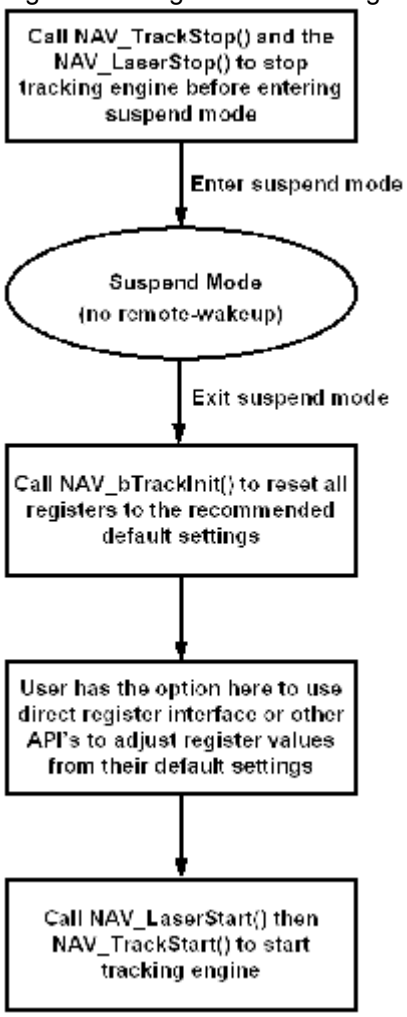


Figure 4. Figure 4. Sensor Enters or Exits Suspend Mode Without Remote-Wakeup

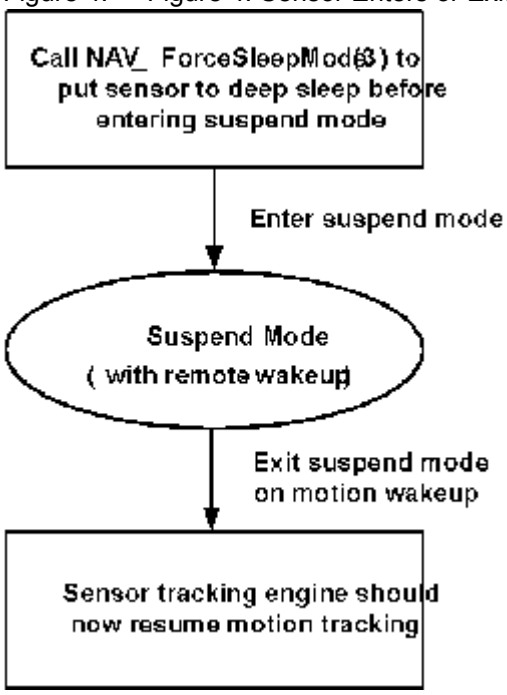
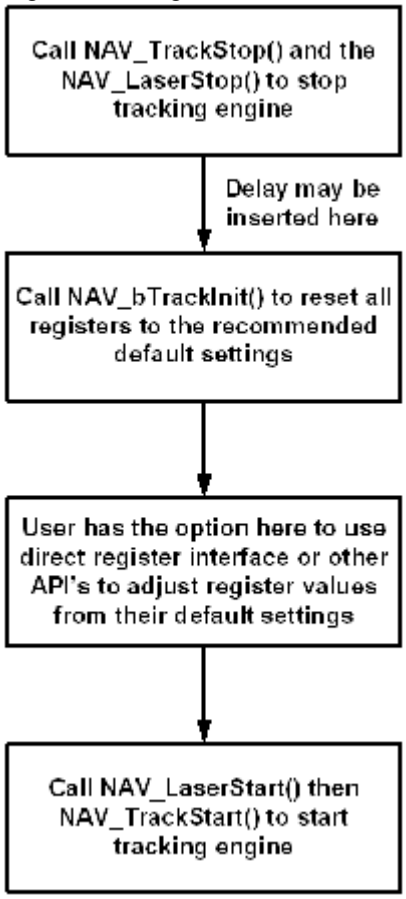


Figure 5. Figure 5. Sensor Enters or Exits Suspend Mode With Remote-Wakeup



Operation Modes

The device has multiple operating modes. These are:

- Active mode with highest speed tracking
  In this mode, the chip tracks the highest speed motion of the mouse. The rate at which the device captures information from the analog chip can be up to 80KHz. The chip also consumes the highest power.
- Active modes with low speed tracking
  In these modes, the device modulates frame rate based on detected speed of mouse motion. This reduces active chip power.
- Sleep Modes

  These are low power modes to enhance battery life. If the device does not detect any motion for a programmable amount of time while in the tracking mode, it transitions to the shallowest sleep modes. The device has been provided with 4 sleep modes. If the chip has been in a sleep mode for long enough time (programmable) without detecting any motion, the chip may enter the next deeper sleep mode if there is one available

  – Shallowest sleep modes with low wakeup time

    This mode is entered when the chip detects no mouse motion for an extended period of time. Once, this mode is entered, the device consumes only leakage power, for a major portion of time. However, the chip needs to detect if mouse movement is happening. Hence, once in a while, the device wakes up the analog super block and takes a few frame samples to check for motion. If no motion is detected, the chip re-enters sleep mode. Else, it moves into track mode.

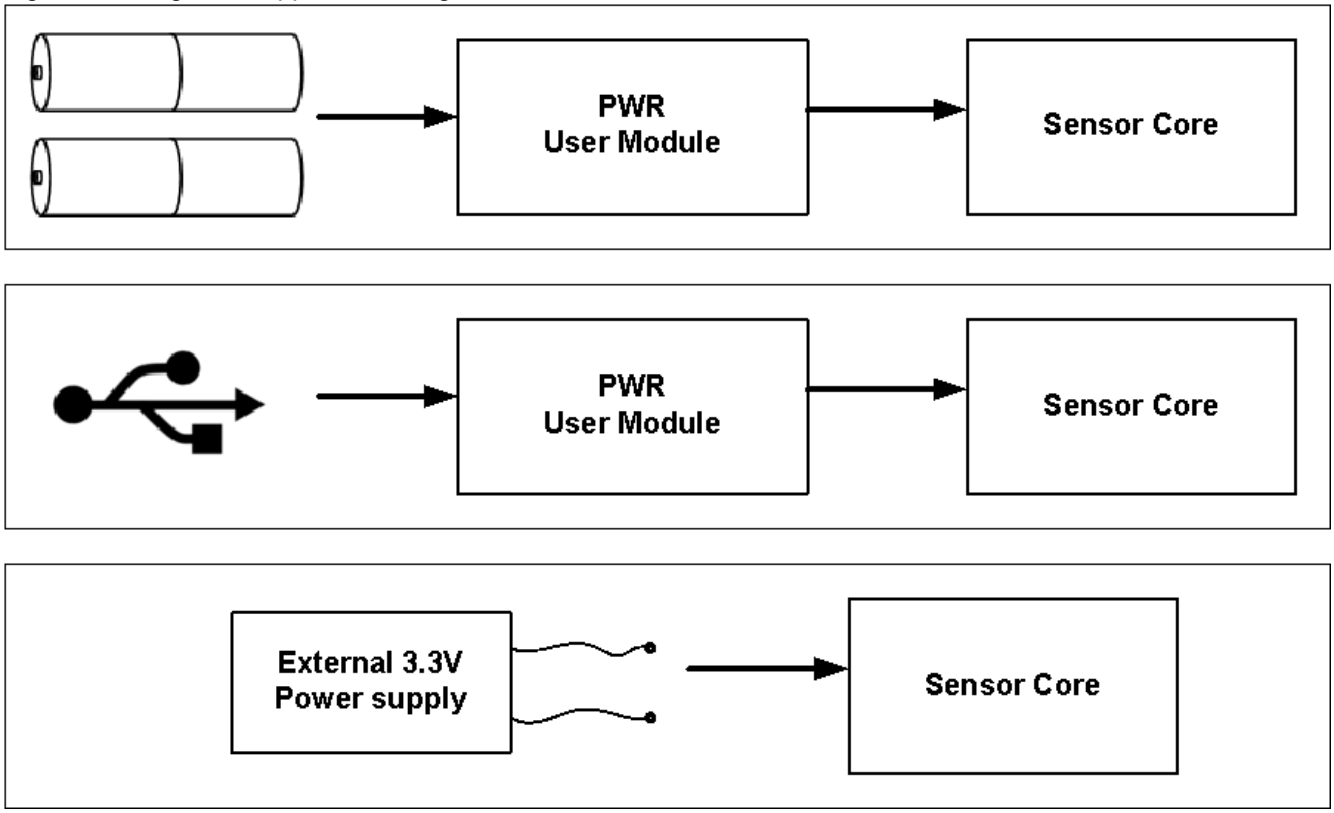
  – Deep sleep modes with higher wake-up times

    These modes are entered when the device detects no mouse motion for a long period of time. There are three such modes. Once a particular deep sleep mode is entered, the chip consumes only leakage power, for a major portion of time. However, the device needs to detect if mouse movement is happening. Hence, once in a while, the chip wakes up the analog superblock and takes a few frame samples to check for motion. If no motion is detected, the device re-enters the same sleep mode. Else, it moves into track mode. This mode differs from the previous mode in the duration between taking frames.

The device can be configured for one of the following:

- Wired (power is supplied by the USB bus)
- Wireless (battery power supply for radio application)
- Dual wired/wireless (battery and USB power are hot swappable)
- External 3.3V supply

Figure 6. Figure 6. Application Diagram for Radio Wireless and Wired USB Mouse Devices



Power System Modes

Wired mode. In this mode, device is powered only from VDD5V. In a typical USB application, this pin is connected to V_{BUS}, the USB 5V power pin. In wired operation, the 3.3V regulator is active. Selection of wired mode in the PWR User Module Parameter area causes the 3.3V regulator to be active by default.

The boost regulator should be off in this configuration. Selection of wired mode in the PWR User Module Parameter area causes the boost regulator to be disabled by default. However, it is the designer's responsibility to ensure that BOOST_GND, BOOST_IN, and BATT signals are connected to DVSS.

Wireless mode. In this mode, the device is powered only from the boost regulator using the BOOST_IN power input. In a typical wireless operation, BOOST_IN is connected to an inductor, BOOST_GND to the battery's negative terminal, and BATT directly to the battery output.

In wireless mode, the boost regulator should be enabled, and the 3.3V regulator should be disabled. Both of these are set by default when wireless mode is selected in the PWR User Module Parameter area. However, for proper operation, the designer must connect VDD5V to DVSS.

External power supply mode. In this mode, both of the internal regulators are disabled, and the chip is powered from a clean, regulated, external supply.

In external mode, both the boost regulator and the 3.3V regulator should be disabled. Selection of external mode in the PWR User Module Parameter area disables the two regulators by default. For proper operation, the designer must connect BOOST_GND, BOOST_IN, BATT, and VDD5V to DVSS.

Wired and Wireless mode.In this mode, the device is allowed to transition between wired and wireless operation, by dynamically changing the configuration of the two regulators in response to which power sources are present.

If both sources are present, then the boost regulator is disabled, and the chip is powered from the VDD5V input. If the VDD5V input is removed, the UM responds to the trip interrupt by enabling the boost regulator. When the VDD5V input is replaced, the UM switches operation back to the 3.3V regulator, to preserve battery life.

Hybrid Operation

When the battery is inserted, the boost turns on. The CPU has an opportunity to monitor the battery insertion and configure the boost through the boost registers to make the boost go into sleep mode. When USB supply is ramped down on detecting the tripper going to 0, core is issued a wakeup interrupt. The boost is switched on and is in the active state. The 1.8V regulator is turned off. Power down signals are asserted to the analog core and analog part enable is deasserted. If the core is in sleep mode it wakes up and provides the 8 MHz clock for digital blocks. The tripper interrupt is asserted and is propagated to core.

When USB supply is ramped up, on detecting the tripper going to 1, the core is issued a wake up interrupt. The 1.8V regulator is turned off.

Figure 7. Power Mode State Diagram for Wired Switchover

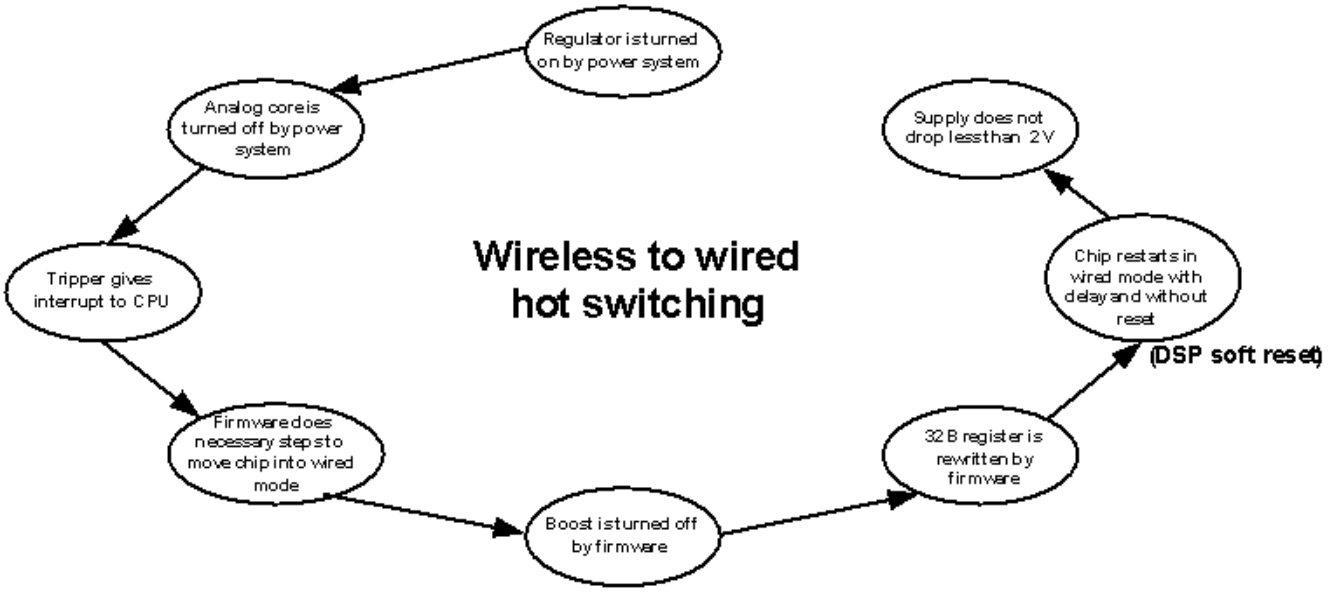
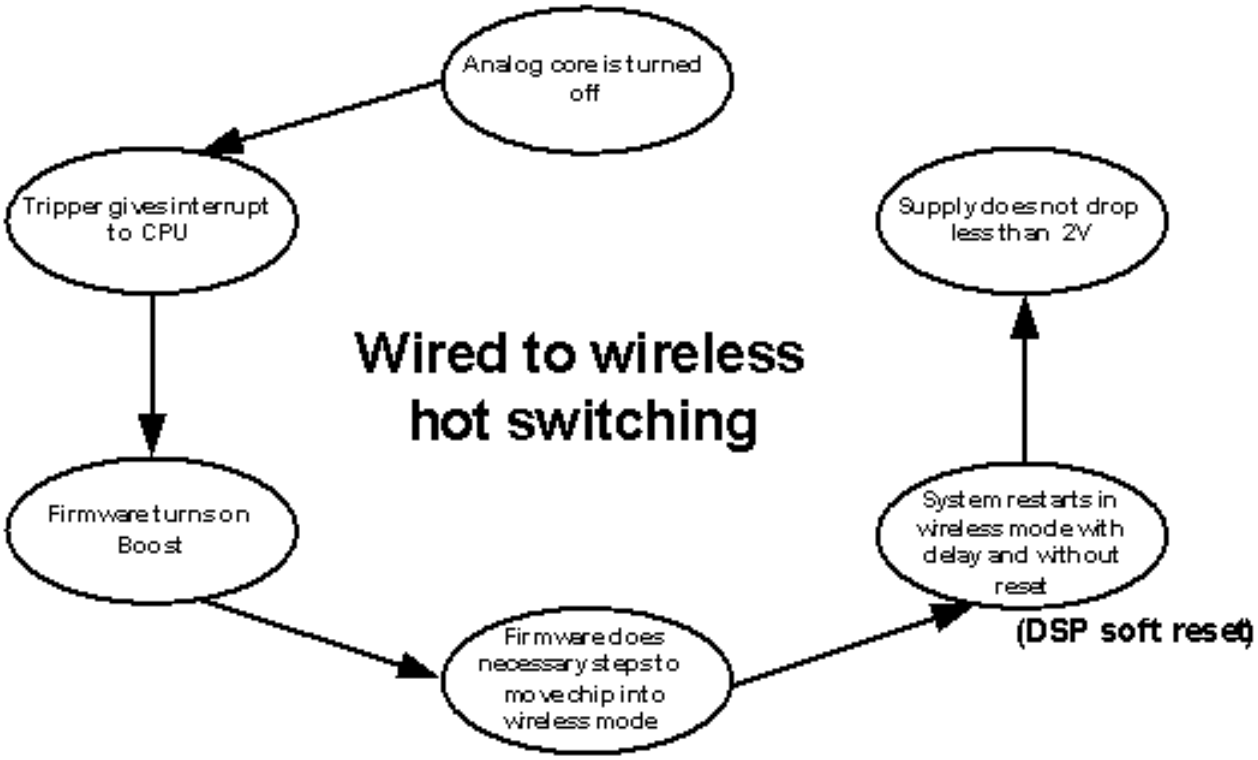


Figure 8. Figure 8. Power Mode State Diagram for Wired Switch



Boost Regulator

The Boost regulator is a device that allows conversion of low voltage (for example, battery) inputs to 3.3V operating levels.This block architecture uses a single inductor and internal rectification to increase or decrease voltage from input to output as required. An inductor is the only additional external component required to implement the converter. External bypass capacitors on Vdd are also required - these capacitors are the same components required when the device is powered by a conventional power supply but are specified more completely to assure proper operation. Rectification is achieved internally rather than requiring additional external Shottky diodes.

3.3V Regulator

This is a linear voltage regulator designed to convert a 5.0V USB supply voltage to 3.3V (in wired applications). The regulator input voltage can vary between 4V and 5.5V. The nominal regulator output is 3.3V. It has been designed such that it never exceeds 3.6V. In wired applications, the Boost circuit is turned off and the regulator supplies the power. The USB pin is used to switch power domain from boost to regulators. When USB power is not supplied in a wireless application, the USB pin is pulled down (1000 kΩ).

Placement

The Optical Navigation User Module can be placed in the dedicated block of CYONS2xxx only.

Parameters and Resources

Navigation Sleep Modes

This sleep modes parameter performs basic power management configuration. Possible values are “enable” or “disable”.

Power System Configuration

This parameter chooses power source and actually sets device configuration. The available choices vary by device. Possible choices are:

Parameter	CYONS2000, CYONS2100, CYONS2010, CYONSTB2010	CYONS2001, CYONS2101, CYONS2011, CYONSTB2011	CYONS2110	CYONSFN2162	CYONSFN2051, CYONSFN2053, CYONSFN2061, CYONSFN2151, CYONSFN2161
5V USB	**• (default)**		**• (default)**	•	
3.3V supply	•	•	•	**• (default)**	**• (default)**
Single cell battery		•	•	•	•
Dual cell battery		**• (default)**	•	•	•

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.

Not all Power management API functions are supported for all devices. The following table defines the applicability of each API for each part number. • - Available on this part.

Table 2. Power management API Applicability vs. Part Number

APIs	CYONS2000, CYONS2100, CYONS2010, CYONSTB2010	CYONS2001, CYONS2101, CYONS2011, CYONSTB2011	CYONS2110, CYONSFN2162
LaserNAV_Start()	•	•	•
LaserNAV_Stop()	•	•	•
LaserNAV_5VRegOn()	•		•
LaserNAV_5VRegOff()	•		•
LaserNAV_BoostOn()		•	•
LaserNAV_BoostOff()		•	•
LaserNAV_5VRegTripEnableInt()			•
LaserNAV_5VRegTripDisableInt()			•
LaserNAV_iReadBattMonitor()		•	•
LaserNAV_BoostUnderVoltageEnableInt()		•	•
LaserNAV_BoostUnderVoltageDisableInt()		•	•
LaserNAV_fHad5VRegTrip()	•		•
LaserNAV_fHadBoostUnderVoltage()	•	•	•
LaserNAV_fCheckTripperOutput()	•		•
LaserNAV_fCheckLvdOutput()	•	•	•
LaserNAV_fCheckInitDone()	•	•	•
LaserNAV_fCheckBoostVoltageReached()		•	•
LaserNAV_EnableUSBSupplyWake()	•		•
LaserNAV_DisableUSBSupplyWake()	•		•
LaserNAV_ShutDown()	•	•	•
LaserNAV_Startup()	•	•	•
LaserNAV_bSetBoostOutput(bOutput)		•	•

Table 3. Power management API Applicability vs. Part Number *(continued)*

APIs	CYONSFN2051	CYONSFN2053, CYONSFN2061, CYONSFN2151, CYONSFN2161
LaserNAV_Start()	•	•
LaserNAV_Stop()	•	•
LaserNAV_5VRegOn()		
LaserNAV_5VRegOff()		
LaserNAV_BoostOn()	•	•
LaserNAV_BoostOff()	•	•
LaserNAV_5VRegTripEnableInt()		
LaserNAV_5VRegTripDisableInt()		
LaserNAV_iReadBattMonitor()	•	•
LaserNAV_BoostUnderVoltageEnableInt()	•	•
LaserNAV_BoostUnderVoltageDisableInt()	•	•
LaserNAV_fHad5VRegTrip()		
LaserNAV_fHadBoostUnderVoltage()	•	•
LaserNAV_fCheckTripperOutput()		
LaserNAV_fCheckLvdOutput()	•	•
LaserNAV_fCheckInitDone()	•	•
LaserNAV_fCheckBoostVoltageReached()	•	•
LaserNAV_EnableUSBSupplyWake()		
LaserNAV_DisableUSBSupplyWake()		
LaserNAV_ShutDown()	•	•
LaserNAV_Startup()	•	•
LaserNAV_bSetBoostOutput(bOutput)	•	

LaserNAV API

Function	Description
General Purpose API	
void **LaserNAV_Start**(void)	Provides initial configuration: turns on boost or 3.3V linear regulator and initializes battery supply.
void **LaserNAV_Stop**(void)	Turns off boost and 3.3V linear regulator.
Power mangement API	
void **LaserNAV_5VRegOn**(void)	Turns on the 5V regulator circuitry.
void **LaserNAV_5VRegOff**(void)	Turns off the 5V regulator circuitry.
void **LaserNAV_BoostOn**(void)	Turns on the boost circuitry.
void **LaserNAV_BoostOff**(void)	Turns off the boost circuitry.
void **LaserNAV_5VRegTripEnableInt**(void)	Enables the 5V regulator trip interrupt.
void **LaserNAV_5VRegTripDisableInt**(void)	Disables the 5V regulator trip interrupt.
INT **LaserNAV_iReadBattMonitor**(void)	Returns scaled DAC code of battery charge state.
void **LaserNAV_BoostUnderVoltageEnableInt**(void)	Enables Boost under-voltage interrupt.
void **LaserNAV_BoostUnderVoltageDisableInt**(void)	Disables Boost under-voltage interrupt.
BOOL **LaserNAV_fHad5VRegTrip**(void)	Checks 5V tripper interrupt status bit. Clears status bit if it is set.
BOOL **LaserNAV_fHadBoostUnderVoltage**(void)	Checks boost under-voltage interrupt status bit. Clears status bit if it is set.
BOOL **LaserNAV_fCheckTripperOutput**(void)	Indicates the filtered tripper output.
BOOL **LaserNAV_fCheckLvdOutput**(void)	Indicates the filtered LVD signal output.
BOOL **LaserNAV_fCheckInitDone**(void)	Indicates the power system startup is done (after a power switching).
BOOL **LaserNAV_fCheckInitDone**(void)	Indicates if initial power up done.
BOOL **LaserNAV_fCheckBoostVoltageReached**(void)	Indicates if battery is absent or is in low level condition.
void **LaserNAV_EnableUSBSupplyWake**(void)	Enables the wakeup to Krypton when USB supply switches.
void **LaserNAV_DisableUSBSupplyWake**(void)	Disables the wakeup to Krypton when USB supply switches.
void **LaserNAV_ShutDown**(void)	Initiates power down sequence of power system.
void **LaserNAV_Startup**(void)	Brings up the power system.
BYTE **LaserNAV_bSetBoostOutput**(BYTE bOutput)	Set the boost output voltage level based on the input given in 0.1V units and the boost output voltage calibration data from FLASH table

Function	Description
Tracking API	
BYTE **LaserNAV_bTrackInit** (void)	Initializes the tracking engine with the built-in register settings.
void **LaserNAV_TrackStart**(void)	Starts the navigation engine.
void **LaserNAV_TrackStop**(void)	Stops the navigation engine.
void **LaserNAV_SetResolution**(WORD wDPI)	Sets desired optical sensor resolution for both X and Y axes
void **LaserNAV_SetXResolution**(WORD wXDPI)	Sets desired optical sensor resolution for both X axis
void **LaserNAV_SetYResolution**(WORD wYDPI)	Sets desired optical sensor resolution for both Y axis
WORD **LaserNAV_wReadResolution**(void)	Returns current resolution. In the axes resolutions are different it returns X axis resolution.
WORD **LaserNAV_wReadXResolution**(void)	Returns current X-axis resolution.
WORD **LaserNAV_wReadYResolution**(void)	Returns current Y-axis resolution.
BOOL **LaserNAV_fReadXYCounts**(POSITION* SXYData)	Reads XY counts and updates the SXYData structure. Returns false if both X and Y are zero, and true otherwise
void **LaserNAV_LiftHeightSetThreshold**(BYTE bLiftThreshold, BYTE bLiftHysteresis)	Change lift height threshold and hysteresis (Beta - not supported).
BYTE **LaserNAV_bLiftHeightReadData**(void)	Returns the current lift height estimation in counts relative to the maximum (Beta - not supported)..
Power-Saving Mode Control API	
void **LaserNAV_ForceSleepMode**(BYTE bSleepMode)	Go to the sleep mode specified by the given index.
void **LaserNAV_ForceTrackMode**(BYTE bTrackMode)	Go to the track mode specified by the given index.
void **LaserNAV_ConfigureSleepMode**(BYTE bSleepMode, WORD wSleepPeriod, WORD wNumSleepPeriods)	Adjust the sleep period before next check-for-motion, and how long to stay in current sleep mode without seeing motion before moving to next deeper sleep mode (number of sleep period repeats), for the sleep mode specified by the given index (0 to 3)
void **LaserNAV_SetSleepDelay**(WORD wDelayMs)	Adjust the delay in mS when transitioning from the lowest tracking mode to the shallowest sleep mode with no motion being detected.
void **LaserNAV_SleepEnableInt**(void)	Enables the sleep interrupt that is generated by the navigation module.
void **LaserNAV_SleepDisableInt**(void)	Disables the sleep interrupt that is generated by the navigation module.
void **LaserNAV_WakeEnableInt**(void)	Enables the wake-up interrupt that is generated by the Navigation module.
void **LaserNAV_WakeDisableInt**(void)	Disables the wake-up interrupt that is generated by the Navigation module.

Function	Description
void **LaserNAV_VCSELErrorEnableInt**(void)	Enables the general interrupt that is generated by the navigation engine as a result of a VCSEL error.
void **LaserNAV_VCSELErrorDisableInt**(void)	Disables the general interrupt that is generated by the navigation engine as a result of a VCSEL error
void **LaserNAV_GlobalEnableInt**(void)	Enables Global interrupt
void **LaserNAV_GlobalDisableInt**(void)	Disables Global interrupt
void **LaserNAV_TrackSleepTransitionEnableInt**(void)	Enables Global subinterrupt: Track/SleepTransition
void **LaserNAV_TrackSleepTransitionDisableInt**(void)	Disables Global subinterrupt: Track/SleepTransition
BOOL **LaserNAV_fHadVcselError**(void)	Checks VCSEL Error status bit
BOOL **LaserNAV_fHadTrackSleepTransition**(void)	Checks Track/Sleep Transition status bit
BOOL **LaserNAV_fHadTrackingBlankout**(void)	Checks Track/Sleep Blackout status bit
void **LaserNAV_ResetVcsel**(void)	Performs DSP soft reset
BYTE **LaserNAV_bReadCurrentPreviousState**(void)	Reads previous and current track/sleep state in sensor
Laser Control API	
void **LaserNAV_LaserStart**(void)	Turns on the laser driver
void **LaserNAV_LaserStop**(void)	Turns off the laser drive
void **LaserNAV_LaserSetPower**(BYTE bPowerSetting)	Manually change the laser power
void **LaserNAV_LaserAGCControl**(BOOL fAGC)	Turn on/off the AGC loop.
void **LaserNAV_LaserModulationControl**(BOOL fOnOff)	Turn on/off the laser modulation.
BYTE **LaserNAV_bLaserReadPower**(void)	Returns the current laser power setting.
BYTE **LaserNAV_bLaserReadEyesafe**(void)	Returns the laser driver eye-safe level for this sensor part.
BYTE **LaserNAV_bLaserReadMaximum**(void)	Returns the laser driver maximum current setting for this sensor part.
void **LaserNAV_LaserSetTestMode**(void)	Sets the laser to CW mode, without modulation and without AGC
void **LaserNAV_LaserSetEyesafe**(BYTE bPowerSetting)	Assigns eye safety VCSEL current settings for all track modes
void **LaserNAV_LaserSetMaximum**(BYTE bPowerSetting)	Assigns operating VCSEL current settings for all track modes

General Purpose API

LaserNAV_Start

Description:

Provides initial configuration: turns on boost or 3.3V linear regulator and initializes battery supply.

C Prototype:

```
void  LaserNAV_Start(void)
```

Assembly:

```
lcall  LaserNAV_Start
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_Stop

Description:

Turns off boost and 3.3V linear regulator.

C Prototype:

```
void  LaserNAV_Stop(void)
```

Assembly:

```
lcall  LaserNAV_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

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

Power Management API

LaserNAV_5VRegOn

Description:

Turns on the 5V regulator circuitry.

C Prototype:

```
void  LaserNAV_5VRegOn(void)
```

Assembly:

```
lcall  LaserNAV_5VRegOn
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_5VRegOff

Description:

Turns off the 5V regulator circuitry.

C Prototype:

```
void  LaserNAV_5VRegOff(void)
```

Assembly:

```
lcall  LaserNAV_5VRegOff
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_BoostOn

Description:

Turns on the boost circuitry.

C Prototype:

```
void LaserNAV_BoostOn(void)
```

Assembly:

```
lcall LaserNAV_BoostOn
```

Parameters:

None

Return Value:

None

Side Effects

See Note ** at the beginning of the API section

LaserNAV_BoostOff

Description:

Turns off the boost circuitry.

C Prototype:

```
void LaserNAV_BoostOff(void)
```

Assembly:

```
lcall LaserNAV_BoostOff
```

Parameters:

None

Return Value:

None

Side Effects

See Note ** at the beginning of the API section

LaserNAV_5VRegTripEnableInt

Description:

Enables the 5V regulator trip interrupt.

C Prototype:

```
void LaserNAV_5VRegTripEnableInt(void)
```

Assembly:

```
lcall LaserNAV_5VRegTripEnableInt
```

Parameter:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_5VRegTripDisableInt

Description:

Disables the 5V regulator trip interrupt .

C Prototype:

```
void LaserNAV_5VRegTripDisableInt(void)
```

Assembly:

```
lcall LaserNAV_5VRegTripDisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_iReadBattMonitor

Description:

Returns scaled DAC code of battery charge state

C Prototype:

```
INT LaserNAV_iReadBattMonitor(void)
```

Assembly:

```
lcall LaserNAV_iReadBattMonitor
;now you can read result through X(MSB) and A(LSB)
```

Parameters:

None

Return Value:

Returns scaled DAC code of battery charge state or corresponding error code through X (MSB) and A (LSB) registers.

Returned Value	Description
-4	DAC code at 0.8V is not defined
-3	DAC code at 1.2V is not defined
-2	DAC code of battery charge estimation is not defined
-1	Zero or negative slope is derived during calculation
[0;32767]	Corresponding code of battery charge state

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_BoostUnderVoltageEnableInt

Description:

Enables Boost under-voltage interrupt.

C Prototype:

```
void LaserNAV_BoostUnderVoltageEnableInt(void)
```

Assembly:

```
lcall LaserNAV_BoostUnderVoltageEnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_BoostUnderVoltageDisableInt

Description:

Disables Boost under-voltage interrupt.

C Prototype:

```
void LaserNAV_BoostUnderVoltageDisableInt(void)
```

Assembly:

```
lcall LaserNAV_BoostUnderVoltageDisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_fHad5VRegTrip

Description:

Checks 5V tripper interrupt status bit. Clears status bit if it is set.

C Prototype:

```
BOOL LaserNAV_fHad5VRegTrip(void)
```

Assembly:

```
lcall LaserNAV_fHad5VRegTrip
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_fHadBoostUnderVoltage

Description:

Checks boost under-voltage interrupt status bit.Clears status bit if it is set.

C Prototype:

```
BOOL LaserNAV_fHadBoostUnderVoltage(void)
```

Assembly:

```
lcall LaserNAV_fHadBoostUnderVoltage
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_fCheckTripperOutput

Description:

Indicates the filtered tripper output.

C Prototype:

```
BOOL LaserNAV_fCheckTripperOutput(void)
```

Assembly:

```
lcall LaserNAV_fCheckTripperOutput
```

Parameters:

None

Return Value:

Returns (through A register) true if USB power is above 3V Tripper trip point, returns false if USB power goes below 3V Tripper point.

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_fCheckLvdOutput

Description:

Indicates the filtered LVD signal output.

C Prototype:

```
BOOL LaserNAV_fCheckLvdOutput(void)
```

Assembly:

```
lcall LaserNAV_fCheckLvdOutput
```

Parameters:

None

Return Value:

Returns (through A register) true if power supply is above LVD trip point, returns false if power is below LVD trip point.

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_fCheckInitDone

Description:

Indicates if initial power up done.

C Prototype:

```
BOOL LaserNAV_fCheckInitDone(void)
```

Assembly:

```
lcall LaserNAV_fCheckInitDone
```

Parameters:

None

Return Value:

Returns (through A register) true if initial power up bit is set false otherwise.

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_fCheckBoostVoltageReached

Description:

Indicates if battery is absent or is in low level condition.

C Prototype:

```
BOOL LaserNAV_fCheckBoostVoltageReached(void)
```

Assembly:

```
lcall LaserNAV_fCheckBoostVoltageReached
```

Parameters:

None

Return Value:

Returns (through A register) true if low battery condition bit is set false otherwise.

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_EnableUSBSupplyWake

Description:

Enables the wakeup to Krypton when USB supply switches.

C Prototype:

```
void LaserNAV_EnableUSBSupplyWake(void)
```

Assembly:

```
lcall LaserNAV_EnableUSBSupplyWake
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_DisableUSBSupplyWake

Description:

Disables the wakeup to Krypton when USB supply switches.

C Prototype:

```
void LaserNAV_DisableUSBSupplyWake(void)
```

Assembly:

```
lcall LaserNAV_DisableUSBSupplyWake
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_ShutDown

Description:

Initiates power down sequence of power system.

C Prototype:

```
void LaserNAV_ShutDown(void)
```

Assembly:

```
lcall LaserNAV_ShutDown
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section

LaserNAV_Startup

Description:

Brings up the power system.

C Prototype:

```
void LaserNAV_Startup(void)
```

Assembly:

```
lcall LaserNAV_Startup
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_bSetBoostOutput

Description:

Set the boost output voltage level based on the input given in 0.1V units and the boost output voltage calibration data from FLASH table2 byte[6].

C Prototype:

```
BYTE LaserNAV_bSetBoostOutput(BYTE bOutput)
```

Assembly:

```
lcall LaserNAV_bSetBoostOutput
```

Parameters:

bOutput - boost output voltage level requested in 0.1V units

Return Value:

Returns 0x00 = success or 0xFF = boost output range error.

Tracking API

LaserNAV_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  LaserNAV_bTrackInit(void)
```

Assembly:

```
lcall  LaserNAV_bTrackInit
;now error code contains is passed through A register
```

Parameters:

None

Return Value:

Return value	Description
0	There was no error during tracking engine initialization
1	Check sum is failed
2	bPowerMode doesn’t correspond to silicon purpose

Side Effects:

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

LaserNAV_TrackStart

Description:

Starts the navigation engine. The LaserNAV_bTrackInit() should be called before calling this function. This gives you the ability to modify tracking registers to your own desired values before you call LaserNAV_TrackStart().

C Prototype:

```
void  LaserNAV_TrackStart(void)
```

Assembly:

```
lcall  LaserNAV_TrackStart
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_TrackStop

Description:

Stops the navigation engine. After the NAV engine is stopped, to restart the engine, call LaserNAV_bTrackInit() first, then you have the option to modify tracking registers to your own desired values, before calling LaserNAV_TrackStart().

C Prototype:

```
void LaserNAV_TrackStop(void)
```

Assembly:

```
lcall LaserNAV_TrackStop
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_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 LaserNAV_SetResolution(WORD wDPI)
```

Assembly:

```
mov   X, >wDPI
mov   A, <wDPI
lcall LaserNAV_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.

LaserNAV_SetXResolution

Description:

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

C Prototype:

```
void LaserNAV_SetXResolution(WORD wXDPI)
```

Assembly:

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

Parameter:

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

Return Value:

None

Side Effects:

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

LaserNAV_SetYResolution

Description:

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

C Prototype:

```
void LaserNAV_SetYResolution(WORD wYDPI)
```

Assembly:

```
mov   X, >wYDPI
mov   A, <wYDPI
lcall LaserNAV_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.

LaserNAV_wReadResolution

Description:

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

C Prototype:

```
WORD LaserNAV_wReadResolution(void)
```

Assembly:

```
lcall LaserNAV_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.

LaserNAV_wReadXResolution

Description:

Returns current X-axis resolution setting in DPI

C Prototype:

```
WORD LaserNAV_wReadXResolution(void)
```

Assembly:

```
lcall LaserNAV_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.

LaserNAV_wReadYResolution

Description:

Returns current Y-axis resolution setting in DPI

C Prototype:

```
WORD LaserNAV_wReadYResolution(void)
```

Assembly:

```
lcall LaserNAV_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.

Power Saving Mode Control API

LaserNAV_ForceSleepMode

Description:

Goes to the sleep mode specified by the given index from 0 to 3 immediately. This function may be called after you call the LaserNAV_TrackStart() function. Sleep mode with higher index uses longer sleep period, that is, implements deeper sleep and saves more power.

C Prototype:

```
void LaserNAV_ForceSleepMode(BYTE bSleepMode)
```

Assembly:

```
mov     A, bSleepMode
lcall LaserNAV_ForceSleepMode
```

Parameters:

bSleepMode - specifies index of the sleep mode to go to immediately. Passed via accumulator.

Return Value:

None

Side Effects:

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

Laser Control API

LaserNAV_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 LaserNAV_LaserSetPower() call.

C Prototype:

```
void  LaserNAV_LaserStart(void)
```

Assembly:

```
lcall  LaserNAV_LaserStart
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_LaserStop

Description:

Turns off the laser driver.

C Prototype:

```
void  LaserNAV_LaserStop(void)
```

Assembly:

```
lcall  LaserNAV_LaserStop
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_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  LaserNAV_VCSELErrorEnableInt(void)
```

Assembly:

```
lcall  LaserNAV_VCSELErrorEnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_VCSELErrorDisableInt

Description:

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

C Prototype:

```
void  LaserNAV_VCSELErrorDisableInt(void)
```

Assembly:

```
lcall  LaserNAV_VCSELErrorDisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_LaserSetPower

Description:

Manually changes the driver current setting. Should only be used when AGC loop is turned off, otherwise there is no effect.

C Prototype:

```
void  LaserNAV_LaserSetPower(BYTE bPowerSetting)
```

Assembly:

```
mov   A, bPowerSetting
lcall  LaserNAV_LaserSetPower
```

Parameters:

bPowerSetting - laser driver current setting

Return Value:

None

Side Effects:

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

LaserNAV_LaserAGCControl

Description:

Turn on/off the AGC loop.

C Prototype:

```
void  LaserNAV_LaserAGCControl(BOOL fAGC)
```

Assembly:

```
mov   A, fAGC
lcall  LaserNAV_LaserAGCControl
```

Parameters:

fAGC - if non-zero value AGC loop on, otherwise AGC loop off

Return Value:

None

Side Effects:

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

LaserNAV_LaserModulationControl

Description:

Turn on or off the laser modulation.

C Prototype:

```
void  LaserNAV_LaserModulationControl(BOOL fOnOff)
```

Assembly:

```
mov   A, fOnOff
lcall  LaserNAV_LaserModulationControl
```

Parameters:

fOnOff - if non-zero value then it takes pulsed mode, otherwise CW mode

Return Value:

None

Side Effects:

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

LaserNAV_bLaserReadPower

Description:

Returns the current laser power setting.

C Prototype:

```
BYTE  LaserNAV_bLaserReadPower(void)
```

Assembly:

```
lcall  LaserNAV_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.

LaserNAV_bLaserReadEyesafe

Description:

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

C Prototype:

```
BYTE  LaserNAV_bLaserReadEyesafe(void)
```

Assembly:

```
lcall  LaserNAV_bLaserReadEyesafe
; 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.

LaserNAV_bLaserReadMaximum

Description:

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

C Prototype:

```
BYTE  LaserNAV_bLaserReadMaximum(void)
```

Assembly:

```
lcall  LaserNAV_bLaserReadMaximum
; 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.

LaserNAV_LaserSetTestMode

Description:

This command sets the laser to CW mode, without modulation and without AGC. These settings are needed by customers to test the laser output power for eye safety. To exit this mode, the user needs to call LaserNAV_LaserStart.

C Prototype:

```
void  LaserNAV_LaserSetTestMode(void)
```

Assembly:

```
lcall  LaserNAV_LaserSetTestMode
```

Parameters:

None

Return Value:

None

Side Effects:

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

LaserNAV_LaserSetEyesafe

Description:

Sets eye safety VCSEL current value.

C Prototype:

```
void  LaserNAV_LaserSetEyesafe(BYTE bEyeSafeCurrent)
```

Assembly:

```
mov A, bEyeSafeCurrent
lcall  LaserNAV_LaserSetEyesafe
```

Parameters:

bEyeSafeCurrent - eye safety VCSEL current settings

Return Value:

None

Side Effects:

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

LaserNAV_LaserSetMaximum

Description:

Sets operating VCSEL current value.

C Prototype:

```
void  LaserNAV_LaserSetMaximum(BYTE bOperatingCurrent)
```

Assembly:

```
mov A, bOperatingCurrent
lcall  LaserNAV_LaserSetMaximum
```

Parameters:

bOperatingCurrent - eye safety VCSEL current settings

Return Value:

None

Side Effects:

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

Sample Firmware Source Code

The C code illustrated here shows you how to use the LaserNAV 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 LaserNAV.h.
//
// typedef struct {
// INT x;
// INT y;
// } POSITION;
//

POSITION XYData;

LaserNAV_Start();
//
// Start the OvationONS II DSP Tracking Mode.
//
// Always call these first three NAV user modules in order.
//
// 1. LaserNAV_bTrackInit()
// 2. LaserNAV_LaserStart()
// 3. LaserNAV_TrackStart()
//

LaserNAV_bTrackInit();
LaserNAV_LaserStart();
LaserNAV_TrackStart();

do
{
//
// Read the change in X and Y counts from the last read.
//
LaserNAV_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 illustrated 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 _LaserNAVInfo
export LaserNAVInfo
_LaserNAVInfo:
LaserNAVInfo: blk 4

area text
export LaserNAV_demo_init
export _LaserNAV_demo_init
export LaserNAV_demo
export _LaserNAV_demo

export _main

_main:

    ; Insert your main assembly code here.
    lcall LaserNAV_demo_init
.loop:
    lcall LaserNAV_demo
jmp .loop
.terminate:
    jmp .terminate


LaserNAV_demo_init:
_LaserNAV_demo_init:

lcall LaserNAV_Start
mov A, LaserNAV_WIRED
lcall LaserNAV_bTrackInit
lcall LaserNAV_LaserStart
lcall LaserNAV_TrackStart

ret

LaserNAV_demo:
```

```
_LaserNAV_demo:
mov X, <LaserNAVInfo
mov A, >LaserNAVInfo
lcall LaserNAV_fReadXYCounts
; data will be located in LaserNAVInfo location
ret
```

The C code illustrated here reads the battery voltage on the VBATT pin. Not all devices support the iReadBattMonitor API.

The C code illustrated here reads the battery voltage on the VBATT pin:

```
    int iBatteryVoltage;

    iBatteryVoltage = LaserNAV_iReadBattMonitor();

// iBatteryVoltage now holds integer value of 100*voltage. For example, if
// the battery voltage is 1.13 volts, iBatteryVoltage will equal 113.
```

The Assembly code illustrated here reads the battery voltage on the VBATT pin. Not all devices support the iReadBattMonitor API.

```
export _main

area bss(RAM)          ; inform assembler that variables follow
iBatteryVoltage: blk 2 ; battery voltage variable
area text
_main:

; Insert your main assembly code here.


LCALL _LaserNAV_iReadBattMonitor
; call function to read battery voltage
; upon return, stack contains high and low bytes of battery voltage * 100. ; For example,
if the battery voltage is 3.06 volts, the scaled battery
; voltage will equal 306. The value at the stack pointer will contain the
; lower byte of this value, ie 0x32, and the value at SP+1 will contain the
; upper byte of this value, ie 0x01.

mov X, SP
mov A, [X]
;LOW byte should be specified as:
mov [iBatteryVoltage+1], A

inc x
mov A, [X]
;HIGH byte of the variable should be specified as:
mov [iBatteryVoltage], A
```

Configuration Registers

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

Table 4. POWER_DOWN_CTRL_REG0, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	Reg33ActEnLv	LVDPdLv	TripPdVrefLv	TripPdLv	Reg18SbyEnLv	Reg18PdXlLv	Reg18ActEnLv	Mode18vReg

Reg33ActEnLV - 3.3V Regulator ON/OFF status. 1- ON; 0 - OFF.

LVDPdLv - Power down signal for Low Voltage Detector block.

TripPdVrefLv - Control signal to power down entire Tripper circuit.

TripPdLv - Control signal to power down only the Comparator part of the Tripper circuit.

Reg18SbyEnLv - Control signal that turns on/off the 1.8V Regulator Standby mode.

Reg18PdXlLv - Control signal to power down the 1.8V Regulator.

Reg18ActEnLv - Control signal that turns on/off the 1.8V.

Mode18vReg - 1.8V mode selection.

Table 5. CONTROL_REGISTER1, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	Reserved			tst0/disc_inh	boost	buck/extmode	clksel	

disc_inh - Shared with tst0, this bit inhibits inductor reverse current detection (inhibits discontinuous operation when set).

boost - This bit enables boost operation.

buck - This bit enables buck operation (not applicable for boost only).

extmode - This bit enables external mode control (ov2s boost only).

clksel - Clock frequency select.

Table 6. INTR_MASK_REG, Bank 2

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

st_forc - This interrupt occurs each time there is a change in the track modes. It is set when track to track or sleep to sleep or sleep to track or track to sleep state transition happens.

boost_uv_init - Boost UV Init Interrupt: A low to high or high to low transition is detected as an interrupt.

trip_fil_out - Whenever power supply switching happens from wireless to wired, this signal toggles from 0 to 1. Whenever power supply switching happens from wired to wireless, this signal toggles from 1 to 0. During both the transitions, interrupt is set.

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

Table 7. SMx_NO_BLKS4TRANS1, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	smx_no_blks4trans (MSB)							

smx_no_blks4trans - total number of blocks before transitioning to deeper sleep mode (MSB)

Table 8. SMx_CHK_INTRVL0, Bank 3

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

smx_chk_intrvl - inactive interval in multiples of 4 ms between two CFM slots (LSB)

Table 9. SMx_CHK_INTRVL1, Bank 3

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

smx_chk_intrvl - inactive interval in multiples of 4 ms between two CFM slots (MSB)

Table 10. TR0_S0_DOWN_SWITCH_DELAY0, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	tr0_s0_down_switch_delay (LSB)							

tr0_s0_down_switch_delay - number of blocks in TR0 before master controller enters sleep (LSB)

Table 11. TR0_S0_DOWN_SWITCH_DELAY1, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	tr0_s0_down_switch_delay (MSB)							

tr0_s0_down_switch_delay - number of blocks in TR0 before master controller enters sleep (MSB)

Table 12. NUM_BLOCKS_IN_BLANKING0, Bank 2

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

num_blocks_in_blanking - number of contiguous blocks for which blanking is detected before it jumps to sleep mode 0 (LSB)

Table 13. NUM_BLOCKS_IN_BLANKING1, Bank 2

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

num_blocks_in_blanking - number of contiguous blocks for which blanking is detected before it jumps to sleep mode 0 (MSB)

Table 14. 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 15. 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 16. 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 17. 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 18. 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 on the fly. These registers can be updated, at any time, but their visibility into the algorithm is synchronized to the next block boundary

Table 19. 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 on the fly. These registers can be updated, at any time, but their visibility into the algorithm is synchronized to the next block boundary

Table 20. 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 on the fly. These registers can be updated, at any time, but their visibility into the algorithm is synchronized to the next block boundary

Table 21. 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 on the fly. These registers can be updated, at any time, but their visibility into the algorithm is synchronized to the next block boundary

Table 22. DISABLE_TRACK_SLEEP, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	Reserved			mode _in_sleep			dis_slsw	dis_tksw

mode_in_sleep - This register is in sync with the T1/T2 programmed values in CFM. The user needs to ensure that "track mode" selected by this 3-bit register (for choosing the appropriate Track mode integration time registers in CFM) are compatible wth the programmed frame rate in CFM. The reg mux uses this 3 -bit register to select the integration times ONLY when sleep-mode indication is 1.

dis_slsw - disables all sleep mode switching. Only track force can be used to switch states if this bit is 1

dis_tksw - disables all types of track mode switching. Only track force can be used to enforce switching if this bit is 1

Table 23. FORCE_STATE_CTRL, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	Reserved			force_state				force_en

force_state - if [0] =1, at the current block boundary, switch to this track mode

force_en - if 1, force state machine to this state. After switch, master controller resets this bit

Table 24. PWR_DSP_CTRL_REG, Bank 1

Bit	7	6	5	4	3	2	1	0
Value	res_cpi_limit_reg				lvd_trim		tripper_calib	

res_cpi_limit_reg - resolution/CPI limiter data

lvd_trim - LVD trimming register

tripper_calib - 3V Tripper Calibration Value

Table 25. THRESHOLD1, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	threshold1							

threshold1 - is the higher threshold used in lift detection logic.

Table 26. THRESHOLD2, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	threshold2							

threshold2 - is the lower threshold used in lift detection logic.

Table 27. INTR_MASK_REG, Bank 2

Bit	7	6	5	4	3	2	1	0
Value	Reserved				StForce	BstUVInit	TripFilOut	vcselErr

StForce - Track/sleep transition interrupt mask (whenever this interrupt is set, it means that there is a change in the track modes. It is set when track to track or sleep to sleep or sleep to track or track to sleep state transition happens).

BstUVInit - Boost UV Init interrupt mask (a low to high or high to low transition is detected as an interrupt).

TripFilOut - Tripper out filter mask (whenever power supply switching happens from wireless to wired, this signal toggles from 0 to 1. Whenever power supply switching happens from wired to wireless, this signal toggles from 1 to 0. During both the transitions, interrupt is set).

vcselErr - VCSEL Error mask (whenever this interrupt is set, it means that VCSEL is ON for more than the desired time interval. This signal makes a transition from 0 to 1 to indicate error).

Table 28. 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 29. 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 eyesafety VCSEL

Table 30. 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 provided to the VCSEL power controller block. This is updated at every block boundry. This is writeable by both CPU and the internal logic. CPU is given the highest priority.

Table 31. GLOBAL_CONFIG_REG0, Bank 2

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

dsp_start - starts tracking engine

Table 32. GLOBAL_CONFIG_REG0, Bank 2

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

DeviceID - choices wired/wireless device configuration

Table 33. 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 34. 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 35. 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 36. 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 37. 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 38. 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 39. 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 40. AVERAGED_AGC_TAR_MAX1, Bank 3

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

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

Table 41. 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 42. 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 43. 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 44. SMx_BLANKING_THRESHOLD_REG, Bank 3

Bit	7	6	5	4	3	2	1	0
Value	sm**x**_blanking_threshold_reg							

sm**x**_blanking_threshold_reg - blanking threshold to be used in corresponding sleep mode CFM. Can be written on the fly.

Table 45. ADC_CTRL_REG, Bank 3

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

sel_hi_bw - When high, makes the bandwidth of diff. TIA from 40 kHz to 150 kHz.

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 for altering the range of the ADC conversion window.

Table 46. 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 47. 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 genera-ted based on the VCSEL_ON signal from the analog block.

Version History

Version	Originator	Description
0.2	DHA	Improved the high speed tracking of CYONS2000, CYONS2001, CYONS2100, CYONS2101, and CYONS2110 PSoC devices.
1.10	DHA	1. Replaced mulu_16x16_32 function by mul_16x16_32. 2. Native paging was restored before mul_16x16_32 call. 3. Some user module variables moved from InterruptRAM to ramTmp and ramCalc RAMs. 4. Corrected label of ReadXYCounts API. 5. Explained limitations of dynamic reconfiguration in the user module 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 © 2009-2012 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.