Features and Overview

- Flexible track and sleep modes
- Configurable lift detection threshold
- Track and sleep modes can be software controlled
- Configurable resolution
- Power settings for eye safe levels of work with laser

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 and lift height.
- 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 semi-automatic 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 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.
Functional Description
The device has multiple operating modes. These are:

Active Modes

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.

**DC and AC Electrical Characteristics**

See the device datasheet for your Ovation device for the electrical characteristics of the Navigation block.

**Placement**

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

**Parameters and Resources**

**SleepModes**

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

**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.

The following tables list the NAV supplied API functions.

<table>
<thead>
<tr>
<th>Table 1. NAV API</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
</tr>
<tr>
<td>General Purpose API</td>
</tr>
<tr>
<td>void NAV_Start(void)</td>
</tr>
<tr>
<td>void NAV_Stop(void)</td>
</tr>
<tr>
<td>Tracking API</td>
</tr>
<tr>
<td>BYTE NAV_bTrackInit (BYTE bPowerMode)</td>
</tr>
<tr>
<td>void NAV.TrackStart(void)</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>void NAV_TrackStop(void)</td>
</tr>
<tr>
<td>void NAV_SetResolution(WORD wDPI)</td>
</tr>
<tr>
<td>void NAV_SetXResolution(WORD wXDPI)</td>
</tr>
<tr>
<td>void NAV_SetYResolution(WORD wYDPI)</td>
</tr>
<tr>
<td>WORD NAV_wReadResolution(void)</td>
</tr>
<tr>
<td>WORD NAV_wReadXResolution(void)</td>
</tr>
<tr>
<td>WORD NAV_wReadYResolution(void)</td>
</tr>
<tr>
<td>BOOL NAV_fReadXYCounts(POSITION* SXYData)</td>
</tr>
<tr>
<td>void NAV_LiftHeightSetThreshold(BYTE bLiftThreshold, BYTE bLiftHysteresis)</td>
</tr>
<tr>
<td>BYTE NAV_bLiftHeightReadData(void)</td>
</tr>
</tbody>
</table>

**Power-Saving Mode Control API**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void NAV_ForceSleepMode(BYTE bSleepMode)</td>
<td>Go to the sleep mode specified by the given index.</td>
</tr>
<tr>
<td>void NAV_ForceTrackMode(BYTE bTrackMode)</td>
<td>Go to the track mode specified by the given index.</td>
</tr>
<tr>
<td>void NAV_ConfigureSleepMode(BYTE bSleepMode, WORD wSleepPeriod, WORD wNumSleepPeriods)</td>
<td>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)</td>
</tr>
<tr>
<td>void NAV_SetSleepDelay(WORD wDelayMs)</td>
<td>Adjust the delay in mS when transitioning from the lowest tracking mode to the shallowest sleep mode with no motion being detected.</td>
</tr>
<tr>
<td>void NAV_SleepEnableInt(void)</td>
<td>Enables the sleep interrupt that is generated by the navigation module.</td>
</tr>
<tr>
<td>void NAV_SleepDisableInt(void)</td>
<td>Disables the sleep interrupt that is generated by the navigation module.</td>
</tr>
<tr>
<td>void NAV_WakeEnableInt(void)</td>
<td>Enables the wake-up interrupt that is generated by the Navigation module.</td>
</tr>
<tr>
<td>void NAV_WakeDisableInt(void)</td>
<td>Disables the wake-up interrupt that is generated by the Navigation module.</td>
</tr>
<tr>
<td>void NAV_VCSELErrorEnableInt(void)</td>
<td>Enables the general interrupt that is generated by the navigation engine as a result of a VCSEL error.</td>
</tr>
<tr>
<td>void NAV_VCSELErrorDisableInt(void)</td>
<td>Disables the general interrupt that is generated by the navigation engine as a result of a VCSEL error</td>
</tr>
<tr>
<td>void NAV_GlobalEnableInt(void)</td>
<td>Enables Global interrupt</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>void NAV_GlobalDisableInt(void)</td>
<td>Disables Global interrupt</td>
</tr>
<tr>
<td>void NAV_TrackSleepTransitionEnableInt(void)</td>
<td>Enables Global subinterrupt: Track/SleepTransition</td>
</tr>
<tr>
<td>void NAV_TrackSleepTransitionDisableInt(void)</td>
<td>Disables Global subinterrupt: Track/SleepTransition</td>
</tr>
<tr>
<td>BOOL NAV_fHadVcselError(void)</td>
<td>Checks VCSEL Error status bit</td>
</tr>
<tr>
<td>BOOL NAV_fHadTrackSleepTransition(void)</td>
<td>Checks Track/Sleep Transition status bit</td>
</tr>
<tr>
<td>BOOL NAV_fHadTrackingBlankout(void)</td>
<td>Checks Track/Sleep Blackout status bit</td>
</tr>
<tr>
<td>void NAV_ResetVcsel(void)</td>
<td>Performs DSP soft reset</td>
</tr>
<tr>
<td>BYTE NAV_bReadCurrentPreviousState(void)</td>
<td>Reads previous and current track/sleep state in sensor</td>
</tr>
</tbody>
</table>

**Laser Control API**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void NAV_LaserStart(void)</td>
<td>Turns on the laser driver</td>
</tr>
<tr>
<td>void NAV_LaserStop(void)</td>
<td>Turns off the laser drive</td>
</tr>
<tr>
<td>void NAV_LaserSetPower(BYTE bPowerSetting)</td>
<td>Manually change the laser power</td>
</tr>
<tr>
<td>void NAV_LaserAGCControl(BOOL fAGC)</td>
<td>Turn on/off the AGC loop.</td>
</tr>
<tr>
<td>void NAV_LaserModulationControl(BOOL fOnOff)</td>
<td>Turn on/off the laser modulation.</td>
</tr>
<tr>
<td>BYTE NAV_bLaserReadPower(void)</td>
<td>Returns the current laser power setting.</td>
</tr>
<tr>
<td>BYTE NAV_bLaserReadEyesafe(void)</td>
<td>Returns the laser driver eye-safe level for this sensor part.</td>
</tr>
<tr>
<td>BYTE NAV_bLaserReadMaximum(void)</td>
<td>Returns the laser driver maximum current setting for this sensor part.</td>
</tr>
<tr>
<td>void NAV_LaserSetTestMode(void)</td>
<td>Sets the laser to CW mode, without modulation and without AGC</td>
</tr>
<tr>
<td>void NAV_LaserSetEyesafe(BYTE bPowerSetting)</td>
<td>Assigns eye safety VCSEL current settings for all track modes</td>
</tr>
<tr>
<td>void NAV_LaserSetMaximum(BYTE bPowerSetting)</td>
<td>Assigns operating VCSEL current settings for all track modes</td>
</tr>
<tr>
<td>void NAV_LaserSetEyesafeMSB(BYTE bPowerSetting)</td>
<td>Sets 3 bits of DAC2 code for VCSEL driver.</td>
</tr>
<tr>
<td>void NAV_LaserSetMaximumMSB(BYTE bPowerSetting)</td>
<td>Sets 3 bits of DAC1 code for VCSEL driver.</td>
</tr>
<tr>
<td>BYTE NAV_bLaserReadEyesafeMSB(void)</td>
<td>Reads 3 bits of DAC2 code for VCSEL driver</td>
</tr>
<tr>
<td>BYTE NAV_bLaserReadMaximumMSB(void)</td>
<td>Reads 3 bits of DAC1 code for VCSEL driver</td>
</tr>
<tr>
<td>void NAV_AnalogStart(void)</td>
<td>Clears the power down bits for AGC, sensor1, sensor2, bicells and ASB</td>
</tr>
<tr>
<td>void NAV_AnalogStop(void)</td>
<td>Sets the power down bits for AGC, sensor1, sensor2, bicells and ASB</td>
</tr>
</tbody>
</table>
Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the NAV_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable and constant symbol. In the following descriptions the instance name has been shortened to NAV for simplicity.

**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 functions may leave A and X unchanged, there is no guarantee they may do so in the future.

**General Purpose API**

**NAV_Start**

**Description:**
A null function, maintained for user module API consistency.

**C Prototype:**

```c
void NAV_Start(void);
```

**Assembly:**

```assembly
lcall NAV_Start
```

**Parameters:**
None

**Return Value:**
None

**Side Effects:**
See Note ** at the beginning of the API section.

**NAV_Stop**

**Description:**
A null function, maintained for user module API consistency.

**C Prototype:**

```c
void NAV_Stop(void);
```

**Assembly:**

```assembly
lcall NAV_Stop
```

**Parameters:**
None

**Return Value:**
None
Side Effects:
See Note ** at the beginning of the API section.

**Tracking API**

*NAV_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 NAV_bTrackInit(BYTE bPowerMode);

**Assembly:**
mov A, bPowerMode
lcall NAV_TrackInit
;now error code contains is passed through A register

**Parameters:**
- bPowerMode - chip power mode

**Return Value:**

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>There was no error during tracking engine initialization</td>
</tr>
<tr>
<td>1</td>
<td>Check sum is failed</td>
</tr>
<tr>
<td>2</td>
<td>bPowerMode doesn’t correspond to silicon purpose</td>
</tr>
</tbody>
</table>

Side Effects:
See Note ** at the beginning of the API section.

*NAV_TrackStart*

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

**C Prototype:**
void NAV_TrackStart(void);

**Assembly:**
lcall NAV_TrackStart

**Parameters:**
- None
Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

**NAV_TrackStop**

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

C Prototype:
void NAV_TrackStop(void);

Assembly:
lcall NAV_TrackStop

Parameters:
None

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

**NAV_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 NAV_SetResolution(WORD wDPI);

Assembly:
mov   X, >wDPI
mov   A, <wDPI
lcall NAV_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.
**NAV_SetXResolution**

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

**C Prototype:**
```c
void NAV_SetXResolution(WORD wXDPI);
```

**Assembly:**
```asm
mov X, >wXDPI
mov A, <wXDPI
lcall NAV_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.

**NAV_SetYResolution**

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

**C Prototype:**
```c
void NAV_SetYResolution(WORD wYDPI);
```

**Assembly:**
```asm
mov X, >wYDPI
mov A, <wYDPI
lcall NAV_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.

**NAV_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 NAV_ReadXResolution. If you set the X and Y resolution with the NAV_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  NAV_wReadResolution(void);
```

**Assembly:**

```
lcall NAV_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.

---

**NAV_wReadXResolution**

**Description:**

Returns current X-axis resolution setting in DPI

**C Prototype:**

```
WORD  NAV_wReadXResolution(void);
```

**Assembly:**

```
lcall NAV_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.

---

**NAV_wReadYResolution**

**Description:**

Returns current Y-axis resolution setting in DPI

**C Prototype:**

```
WORD  NAV_wReadYResolution(void);
```

**Assembly:**

```
lcall NAV_wReadYResolution
; now X contains MSB and A - LSB of returned value
```
Parameters:

None

Return Value:

Returns the current y-axis resolution setting in DPI.

\[X <= \text{MSB part of result}; A <= \text{LSB part of result}\]

Side Effects:

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

**NAV_fReadXYCounts**

Description:

Reads XY counts and updates the SXYData structure. Upon reading, the XY counts register is cleared. This function assumes the following typedef:

```c
typedef struct {INT iX; INT iY;} POSITION;
```

and POSITION type is supported by NAV user module.

C Prototype:

```c
BOOL NAV_fReadXYCounts(POSITION* SXYData);
```

Assembly:

```assembly
mov X, <SXYData
mov A, >SXYData
lcall NAV_fReadXYCounts
; now A contains returned value
```

Parameters:

Pointer MSB => A, LSB => X that contains tracking data after function returns

Return Value:

\[A <= \text{Returns true if non-zero xy_data was read, false if xy data is zero}\]

Side Effects:

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

The XY registers of optical navigation system is cleared after this function call. Function modifies value of structure that SXYData points on.

**NAV_LiftHeightSetThreshold**

Description:

Changes lift height threshold and hysteresis

C Prototype:

```c
void NAV_LiftHeightSetThreshold(BYTE bLiftThreshold, BYTE bLiftHysteresis);
```

Assembly:

```assembly
mov X, bLiftHysteresis
mov A, bLiftThreshold
lcall NAV_LiftHeightSetThreshold
```
Parameters:
   A <= bLiftThreshold (Lift height threshold in relative units where the tracking is cut off. Allowable range is 0 - 7).
   X <= bLiftHysteresis (Lift height hysteresis in relative units to stabilize lift detection. bLiftHysteresis uses the same units and can range from 0 (no hysteresis) to bLiftThreshold).

Return Value:
   None

Side Effects:
   See Note ** at the beginning of the API section.

**NAV_bLiftHeightReadData**

Description:
   Returns the current lift height estimation in counts relative to the maximum. Range is 0 - 7 counts.

C Prototype:
   BYTE NAV_bLiftHeightReadData(void);

Assembly:
   lcall NAV_bLiftHeightReadData
   ; now A contains returned value

Parameters:
   None

Return Value:
   A <= current lift height estimation

Side Effects:
   See Note ** at the beginning of the API section.

**Power-Saving Mode Control API**

**NAV_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 NAV_TrackStart() function. Sleep mode with higher index uses longer sleep period, that is, implements deeper sleep and saves more power.

C Prototype:
   void NAV_ForceSleepMode(BYTE bSleepMode);

Assembly:
   mov     A, bSleepMode
   lcall    NAV_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.

**NAV_ForceTrackMode**

Description:
Jumps to the tracking mode specified by the given index from 0 to 5 immediately. This function may be called only after the NAV_TrackStart() function. Tracking mode with higher index uses higher sampling rate.

**C Prototype:**
void NAV_ForceTrackMode(BYTE bTrackMode);

**Assembly:**
mov     A, bTrackMode
lcall    NAV_ForceTrackMode

Parameters:
bTrackMode - specifies index of the tracking mode to jump to immediately. Passed via accumulator.

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

**NAV_ConfigureSleepMode**

Description:
Sets the sleep period (before next check-for-motion), sleep mode duration (without seeing motion before moving to next deeper sleep mode) and number of sleep period repeats for the sleep mode specified by the given index (0 to 3). This function should be called after NAV_TrackInit() function (initializes tracking engine) and before calling the NAV_TrackStart() function (starting tracking engine). Please note that sleep mode with higher index should use longer sleep period, that is, implements deeper sleep and saves more power. Sleep mode 0 is the shallowest sleep mode; sleep mode 3 is the deepest sleep mode.

**C Prototype:**
void NAV_ConfigureSleepMode(BYTE bSleepMode, WORD wSleepPeriod, WORD wNumSleepPeriods);

**Assembly:**
mov     A, >wNumSleepPeriods       ;MSB of last argument
push    A
mov     A, <wNumSleepPeriods       ;LSB
push    A
mov     A, >wSleepPeriod           ;MSB
push    A
mov     A, <wSleepPeriod           ;LSB
push A
mov A, bSleepMode ;first argument
push A
lcall NAV_ConfigureSleepMode
add SP, -5 ;stack release

Parameters:

bSleepMode - specifies the index of the sleep mode to adjust.
wSleepPeriod - sleep period in ms before next check-for-motion activity.
wNumSleepPeriods - number of sleep period repeats before transitioning to next deeper sleep mode

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

NAV_SetSleepDelay

Description:
Adjusts the delay in ms when transitioning from the lowest tracking mode (track mode 0) to the shallowest sleep mode (sleep mode 0) with no motion being detected. This function must be called after NAV_Trackinit() and before calling the NAV_TrackStart() function (starting tracking engine).

C Prototype:
void NAV_SetSleepDelay(WORD wDelayMs);

Assembly:
mov X, >wDelayMs
mov A, <wDelayMs
lcall NAV_wDelayMs

Parameters:
wDelayMs - delay in ms before transitioning from tracking mode to sleep mode when no motion is detected (X <= MSB; A <= LSB)

Return Value:
None.

Side Effects:
See Note ** at the beginning of the API section.

NAV_SleepEnableInt

Description:
Enables the sleep interrupt that is generated by the Navigation module. In typical usage this interrupt is used by the NAV user module to tell the CPU core to go to sleep

C Prototype:
void NAV_SleepEnableInt(void);
Assembly:
  lcall NAV_SleepEnableInt

Parameters:
  None

Return Value:
  None

Side Effects:
  See Note ** at the beginning of the API section.

**NAV_SleepDisableInt**

Description:
  Disables the sleep interrupt that is generated by the Navigation module.

C Prototype:
  void NAV_SleepDisableInt(void);

Assembly:
  lcall NAV_SleepDisableInt

Parameters:
  None

Return Value:
  None

Side Effects:
  See Note ** at the beginning of the API section.

**NAV_WakeEnableInt**

Description:
  Enables the wake-up interrupt that is generated by the Navigation module. In typical usage this interrupt is used by the Nav module to tell the Krypton controller to wake from sleep, which may occur if the mouse is configured to support wake-on-motion from a USB suspend.

C Prototype:
  void NAV_WakeEnableInt(void);

Assembly:
  lcall NAV_WakeEnableInt

Parameters:
  None

Return Value:
  None
Side Effects:
    See Note ** at the beginning of the API section.

*NAV_WakeDisableInt*

Description:
    Disables the wake-up interrupt that is generated by the Navigation module.

C Prototype:
    void NAV_WakeDisableInt(void);

Assembly:
    lcall NAV_WakeDisableInt

Parameters:
    None

Return Value:
    None

Side Effects:
    See Note ** at the beginning of the API section.

*NAV_GlobalEnableInt*

Description:
    Enables global interrupt.

C Prototype:
    void NAV_GlobalEnableInt(void);

Assembly:
    lcall NAV_GlobalEnableInt

Parameters:
    None

Return Value:
    None

Side Effects:
    See Note ** at the beginning of the API section.

*NAV_GlobalDisableInt*

Description:
    Disables Global interrupt.

C Prototype:
    void NAV_GlobalDisableInt(void);
Assembly:
  lcall  NAV_GlobalDisableInt

Parameters:
  None

Return Value:
  None

Side Effects:
  See Note ** at the beginning of the API section.

**NAV_TrackSleepTransitionEnableInt**

Description:
  Enables Track/SleepTransition interrupt.

C Prototype:
  void  NAV_TrackSleepTransitionEnableInt(void);

Assembly:
  lcall  NAV_TrackSleepTransitionEnableInt

Parameters:
  None

Return Value:
  None

Side Effects:
  See Note ** at the beginning of the API section.

**NAV_TrackSleepTransitionDisableInt**

Description:
  Disables Track/SleepTransition interrupt.

C Prototype:
  void  NAV_TrackSleepTransitionDisableInt(void);

Assembly:
  lcall  NAV_TrackSleepTransitionDisableInt

Parameters:
  None

Return Value:
  None

Side Effects:
  See Note ** at the beginning of the API section.
**NAV_fHadVcselError**

**Description:**
Checks VCSEL Error status bit. Clears if set.

**C Prototype:**
```c
BOOL NAV_fHadVcselError(void);
```

**Assembly:**
```assembly
lcall NAV_fHadVcselError
```

**Parameters:**
None

**Return Value:**
Returns (through A register) VCSEL Error status bit

**Side Effects:**
See Note ** at the beginning of the API section.

**NAV_fHadTrackSleepTransition**

**Description:**
Checks Track/Sleep Transition status bit. Clears if set.

**C Prototype:**
```c
BOOL NAV_fHadTrackSleepTransition(void);
```

**Assembly:**
```assembly
lcall NAV_fHadTrackSleepTransition
```

**Parameters:**
None

**Return Value:**
Returns (through A register) Track/Sleep Transition status bit

**Side Effects:**
See Note ** at the beginning of the API section.

**NAV_fHadTrackingBlankout**

**Description:**
Checks Blankout status bit. Interrupt occurs when signal is too low and tracking output is blanked out.

**C Prototype:**
```c
BOOL NAV_fHadTrackingBlankout(void);
```

**Assembly:**
```assembly
lcall NAV_fHadTrackingBlankout
```
Laser Control API

**Parameters:**
None

**Return Value:**
Returns (through A register) Blankout status bit

**Side Effects:**
See Note ** at the beginning of the API section.

*NAV_ResetVcsel*

**Description:**
Performs ASB and DSP soft reset.

**C Prototype:**

```c
void NAV_ResetVcsel(void);
```

**Assembly:**
```
lcall NAV_ResetVcsel
```

**Parameters:**
None

**Return Value:**
None

**Side Effects:**
See Note ** at the beginning of the API section.

*NAV_bReadCurrentPreviousState*

**Description:**
Reads current and previous track/sleep state in sensor.

**C Prototype:**
```
BYTE NAV_bReadCurrentPreviousState(void);
```

**Assembly:**
```
lcall NAV_bReadCurrentPreviousState
```

**Parameters:**
None

**Return Value:**
Returns (through A register) current (7:4 bits) and previous (3:0 bits) track/sleep states.

**Side Effects:**
See Note ** at the beginning of the API section.

Laser Control API
NAV_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 NAV_LaserSetPower() call.

C Prototype:
void NAV_LaserStart(void);

Assembly:
lcall NAV_LaserStart

Parameters:
None

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

NAV_LaserStop

Description:
Turns off the laser driver.

C Prototype:
void NAV_LaserStop(void);

Assembly:
lcall NAV_LaserStop

Parameters:
None

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

NAV_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 NAV_VCSELErrorEnableInt(void);
Assembly:

lcall NAV_VCSELErrorEnableInt

Parameters:
None

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

---

**NAV_VCSELErrorDisableInt**

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

C Prototype:

```c
void NAV_VCSELErrorDisableInt(void);
```

Assembly:

lcall NAV_VCSELErrorDisableInt

Parameters:
None

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

---

**NAV_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:

```c
void NAV_LaserSetPower(BYTE bPowerSetting);
```

Assembly:

```assembly
mov A, bPowerSetting
lcall NAV_LaserSetPower
```

Parameters:
- bPowerSetting - laser driver current setting

Return Value:
None
Side Effects:
See Note ** at the beginning of the API section.

**NAV_LaserAGCControl**

**Description:**
Turn on/off the AGC loop.

**C Prototype:**
void NAV_LaserAGCControl(BOOL fAGC);

**Assembly:**
mov A, fAGC
lcall NAV_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.

**NAV_LaserModulationControl**

**Description:**
Turn on or off the laser modulation.

**C Prototype:**
void NAV_LaserModulationControl(BOOL fOnOff);

**Assembly:**
mov A, fOnOff
lcall NAV_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.

**NAV_bLaserReadPower**

**Description:**
Returns the current laser power setting.

**C Prototype:**
BYTE NAV_bLaserReadPower(void);
Assembly:
lcall NAV_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.

**NAV_bLaserReadEyesafe**

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

C Prototype:
BYTE NAV_bLaserReadEyesafe(void);

Assembly:
lcall NAV_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.

**NAV_bLaserReadMaximum**

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

C Prototype:
BYTE NAV_bLaserReadMaximum(void);

Assembly:
lcall NAV_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.

**NAV_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 NAV_LaserStart.

**C Prototype:**

```c
void NAV_LaserSetTestMode(void);
```

**Assembly:**

```assembly
lcall NAV_LaserSetTestMode
```

**Parameters:**
None

**Return Value:**
None

**Side Effects:**
See Note ** at the beginning of the API section.

**NAV_LaserSetEyesafe**

Description:
Sets eye safety VCSEL current value.

**C Prototype:**

```c
void NAV_LaserSetEyesafe(BYTE bEyeSafeCurrent);
```

**Assembly:**

```assembly
mov A, bEyeSafeCurrent
lcall NAV_LaserSetEyesafe
```

**Parameters:**

```
bEyeSafeCurrent - eye safety VCSEL current settings
```

**Return Value:**
None

**Side Effects:**
See Note ** at the beginning of the API section.

**NAV_LaserSetMaximum**

Description:
Sets operating VCSEL current value.
C Prototype:

```c
void NAV_LaserSetMaximum(BYTE bOperatingCurrent);
```

Assembly:

```assembly
mov A, bOperatingCurrent
lcall NAV_LaserSetMaximum
```

Parameters:

- `bOperatingCurrent` - eye safety VCSEL current settings

Return Value:

None

Side Effects:

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

---

**NAV_LaserSetEyesafeMSB**

Description:

Sets 3 bits of DAC2 code for VCSEL driver (this is used as MSB 3 bits for the eye safety current provided to VCSEL driver).

C Prototype:

```c
void NAV_LaserSetEyesafeMSB(BYTE bEyesafeCurrCode);
```

Assembly:

```assembly
mov A, bEyesafeCurrCode
lcall NAV_LaserSetEyesafeMSB
```

Parameters:

- `bEyesafeCurrCode` - eye safety current code (000b - highest current; 111b - lowest current)

Return Value:

None

Side Effects:

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

---

**NAV_LaserSetMaximumMSB**

Description:

Sets 3 bits of DAC1 code for VCSEL driver (this is used as MSB 3 bits for the operating current provided to VCSEL driver).

C Prototype:

```c
void NAV_LaserSetMaximumMSB(BYTE bOperatingCurrCode);
```

Assembly:

```assembly
mov A, bOperatingCurrCode
lcall NAV_LaserSetMaximumMSB
```

Parameters:

- `bOperatingCurrCode` - operating current code (000b - highest current; 111b - lowest current)
Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

**NAV_bLaserReadEyesafeMSB**

Description:
Reads 3 bits of DAC2 code for VCSEL driver.

C Prototype:
```c
BYTE NAV_bLaserReadEyesafeMSB(void);
```

Assembly:
```assembly
lcall  NAV_bLaserReadEyesafeMSB
```

Parameters:
None

Return Value:
Returns (through A register) DAC2 code for VCSEL eyesafe current (000b - highest current; 111b - lowest current)

Side Effects:
See Note ** at the beginning of the API section.

**NAV_bLaserReadMaximumMSB**

Description:
Reads 3 bits of DAC1 code for VCSEL driver.

C Prototype:
```c
BYTE NAV_bLaserReadMaximum(void);
```

Assembly:
```assembly
lcall  NAV_bLaserReadMaximum
```

Parameters:
None

Return Value:
Returns (through A register) DAC1 code for VCSEL operating current (000b - highest current; 111b - lowest current)

Side Effects:
See Note ** at the beginning of the API section.

**NAV_AnalogStart**

Description:
Clears the power down bits for AGC, sensor1, sensor2, bicells and ASB.
C Prototype:
void NAV_AnalogStart(void);

Assembly:
lcall NAV_AnalogStart

Parameters:
None

Return Value:
None

Side Effects:
See Note ** at the beginning of the API section.

**NAV_AnalogStop**

Description:
Sets the power down bits for AGC, sensor1, sensor2, bicells and ASB.

C Prototype:
void NAV_AnalogStop(void);

Assembly:
lcall NAV_AnalogStop

Parameters:
None

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 NAV User Module.

#define endless_loop 1
#define WIRED 0

// Including the user module header file for POSITION definition.
#include "NAV.h"

void main(void)
{

    // Integers to hold our X an Y counts read from the sensor.
    //
    int iX, iY;
/ POSITION is defined in NAV.h. 

typedef struct {  
  INT x;  
  INT y;  
} POSITION;

POSITION XYData;

// Start the OvationONS II DSP Tracking Mode.  
// Always call these first three NAV user modules in order.  
// 1. NAV_TrackInit()  
// 2. NAV_LaserStart()  
// 3. NAV_TrackStart()  
//

NAV_bTrackInit(WIRED);  
NAV_LaserStart();  
NAV_TrackStart();

do
{
  // Read the change in X and Y counts from the last read.
  
  NAV_fReadXYCounts(&XYData);

  // pXYData now points to X and Y movement 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 similar function 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 _NavInfo
export NavInfo
_NavInfo:
NavInfo: blk 4

area text
export nav_demo_init
export _nav_demo_init
export nav_demo
export _nav_demo

export _main

_main:
    ; Insert your main assembly code here.
    lcall nav_demo_init
    .loop:
        lcall nav_demo
    jmp .loop
    .terminate:
        jmp .terminate

nav_demo_init:
    _nav_demo_init:

    mov A, NAV_1_WIRED
    lcall NAV_1_BTrackInit
    lcall NAV_1_LaserStart
    lcall NAV_1_TrackStart
    ret

    nav_demo:
    _nav_demo:
    mov X, <NavInfo
    mov A, >NavInfo
    lcall NAV_1_fReadXYCounts
    ; data will be located in NavInfo location
    ret

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 2.  SMx_NO_BLKS4TRANS0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>smx_no_blks4trans (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
smx_no_blks4trans - total number of blocks before transitioning to deeper sleep mode (LSB)

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>smx_no_blks4trans (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>smx_no_blks4trans (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>smx_chk_intrvl (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>smx_chk_intrvl (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>tr0_s0_down_switch_delay (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>tr0_s0_down_switch_delay (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>num_blocks_in_blanking (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>num_blocks_in_blanking (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x_cnt_reg_buf - X count buffer register (LSB)

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>x_cnt_reg_buf (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 11. X_CNT_REG_BUF1, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>x_cnt_reg_buf (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x_cnt_reg_buf - X count buffer register (MSB)

Table 12. Y_CNT_REG_BUF0, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>y_cnt_reg_buf (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

y_cnt_reg_buf - X count buffer register (LSB)

Table 13. Y_CNT_REG_BUF1, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>y_cnt_reg_buf (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

y_cnt_reg_buf - X count buffer register (MSB)

Table 14. RES_SCAL_DX0, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>res_scal_dx (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 15. RES_SCAL_DX1, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>res_scal_dx (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 16. RES_SCAL_DY0, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>res_scal_dy (LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 17. RES_SCAL_DY1, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>res_scal_dy (MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 18. DISABLE_TRACK_SLEEP, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Reserved</td>
<td>mode_in_sleep</td>
<td>dis_slsw</td>
<td>dis_tksw</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 with 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 19. FORCE_STATE_CTRL, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Reserved</td>
<td>force_state</td>
<td>force_en</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 20. PWR_DSP_CTRL_REG, Bank 1

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>res_cpi_limit_reg</td>
<td>lvd_trim</td>
<td>tripper_calib</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

res_cpi_limit_reg - resolution/CPI limiter data

lvd_trim - LVD trimming register

tripper_calib - 3V Tripper Calibration Value

Table 21. THRESHOLD1, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>threshold1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 22. THRESHOLD2, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>threshold2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 23. INTR_MASK_REG, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Reserved</td>
<td>StForce</td>
<td>BstUVInit</td>
<td>TripFilOut</td>
<td>vcselErr</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 24. INT_MSK3, Bank 0

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Reserved</td>
<td>glb_ons</td>
<td>wkp_ons</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

glb_ons - mask optical navigation system global interrupt

wkp_ons - mask optical navigation system wakeup interrupt

Table 25. EYE_CW_VCSEL_CUR0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>eye_cw_vcsel_cur</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

eye_cw_vcsel_cur0 - continuous wave eyesafety VCSEL

Table 26. VCSEL_DAC_CURRENT_AG, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>vcsel_dac_current_agc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Table 27. GLOBAL_CONFIG_REG0, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Reserved</td>
<td>dsp_start</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

dsp_start - starts tracking engine

Table 28. GLOBAL_CONFIG_REG0, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>DeviceID</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DeviceID - choices wired/wireless device configuration

Table 29. BG_BUFF_LVD_TRIM, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Reserved</td>
<td>LVD_offset_lv_trim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LVD_offset_lv_trim - LVD trimming setting

Table 30. TMx_EYE_SAFE_CURR0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>tmx_eyesaf_curr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

tmx_eyesaf_curr - Eye safety current for corresponding track mode
### Table 31. TMx_MAX_VCSEL_PWR0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>tmx_max_vcsel_pwr</td>
</tr>
</tbody>
</table>

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

### Table 32. MAX_CW_VCSEL_CUR0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>max_cw_vcsel_cur</td>
</tr>
</tbody>
</table>

max_cw_vcsel_cur0 - continuous wave Max VCSEL

### Table 33. AVERAGED_AGC_TAR_MIN0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>averaged_agc_tar_min (LSB)</td>
</tr>
</tbody>
</table>

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

### Table 34. AVERAGED_AGC_TAR_MIN1, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 35. AVERAGED_AGC_TAR_MAX0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>averaged_agc_tar_max (LSB)</td>
</tr>
</tbody>
</table>

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

### Table 36. AVERAGED_AGC_TAR_MAX1, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 37. PEER_PRESSURE_PIPELINE_DEPTH, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>peer_pressure_pipeline_depth</td>
</tr>
</tbody>
</table>

peer_pressure_pipeline_depth - peer pressure depth to be used. If this is programmed to 0, peer pressure is disabled.
Table 38. TMx_BLNK_HYST, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>tmx_blnk_hyst</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

tmx_blnk_hyst - blanking hysteresis value for each tracking mode

Table 39. TMx_BLNK_THRES, Bank 2

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>tmx_blnk_thres</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

tmx_blnk_thres - blanking threshold value for each tracking mode (blank out signal if dpp < Blank Thrsh)

Table 40. SMx_BLANKING_THRESHOLD_REG, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>smx_blanking_threshold_reg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
smx_blanking_threshold_reg - blanking threshold to be used in corresponding sleep mode CFM. Can be written on the fly.

Table 41. ADC_CTRL_REG, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Reserved</td>
<td>sel_hi_bw</td>
<td>vbias_en</td>
<td>winshift_hw</td>
<td>winalt_hw</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
sel_hi_bw - When high, makes the bandwidth of diff. TIA from 40kHz to 150kHz

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

winshift_hw - This register is for shifting the full ADC conversion window

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

Table 42. CURRENT_VCSEL_OPPT_REG0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>current_vcsel_oppt_reg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
current_vcsel_oppt_reg - Current VCSEL operating current value. These registers are generated based on the VCSEL_ON signal from the analog block

Table 43. CURRENT_VCSEL_EYESAF_REG0, Bank 3

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>current_vcsel_eyesaf_reg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Version</th>
<th>Originator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>DHA</td>
<td>Added Version History</td>
</tr>
</tbody>
</table>

**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.