A320 Optical Finger Navigation Sensor

Application Note 5423

1.0 Introduction

Avago Technologies has introduced a family of low profile Optical Finger Navigation (OFN) sensors that are suitable as navigational input devices for mobile and handheld applications. These OFN sensors are integrated with an IR LED (InfraRed Light Emitting Diode) and have low-power consumption. Coupled with their low-profile, small form factor packaging, these sensors present an excellent solution for hand-held devices requiring navigation capabilities. Examples of applications include integrated input devices, battery-powered input devices and finger input devices. These sensors also incorporate a slew of optical finger navigation features that are specifically designed to heighten, improve and enhance the user experience.

The purpose of this application notes is to help the users with their own firmware development. The users are encouraged to refer to the algorithm described in this document and proceed to develop their own firmware or port the firmware C codes that Avago has developed for the OFN demo to suit their applications.

This document is divided into eleven main sections. Section one serves as an introduction. Section two describes the key register that the user firmware can access to configure and control the OFN sensors. Section three to eight describe each of the OFN registers. Section nine provides the details of the demo firmware implementation including the power-up sequence, power saving consideration, software routines, operating modes and software FPD (Finger Presence Detection) algorithm. The tenth section gives a detailed introduction about Pixel Grab function of OFN sensor. The last section regards the OFN power up supply reference design.

2.0 The OFN Registers

There are five major features that are built into the OFN sensor. They are:

- Speed Switching (SS) – allows the cursor movement to be automatically adjusted according to user's finger speed
- Assert or De-assert (A/D) – allows cursor sensitivity control during user finger lift on and off the sensor
- Finger Presence Detection (FPD) – determines whether user's finger is on sensor
- XY Quantization (XYQ) – allows motion reporting in quantized format for scroll and rocker emulation
- XY Scaling (XYS) – allows motion reporting for X and Y axis to be doubled or inverted for use in portrait or landscape mode.

In order to activate these features, it is advisable to follow the steps in the "Notes on Power Up" for initialization described in section 9.4. The OFN engine must be turned on by setting the most significant bit of register address 0x60. Other features can then be set individually. Alternatively a single write value of 0xE4 to register 0x60 will enable OFN engine, Speed Switching, Assert/De-assert and Finger Presence Detect. XY Quantization and XY Scaling are typically not used in normal finger navigation mode. These features are intended for rocker switch emulation.

OFN_Engine Address: 0x60
Access: Read/Write Reset Value: 0x00

Bit	7	6	5	4	3	2	1	0
Field	Engine	Speed	Assert/ Deassert	XY Q	Reserved	Finger	XY_Scale	Reserved

Data Type: Bit field

USAGE: This register is used to set several properties of the sensor.

Table 1. A320 OFN Engine Register

Field Name	**Description**
Engine	**Sets optical finger navigation properties.** 0 = Disable properties 1 = Enable properties
Speed	Sets speed switching 0 = Enable manual speed switching 1 = Enable automatic speed switching
Assert/Deassert	Sets Assert/Deassert mode 0 = Disable Assert/Deassert 1 = Enable Assert/Deassert
XY Q	Sets XY quantization 0 = Disable quantization 1 = Enable quantization
Finger	Sets finger present detection 0 = Disable finger present detection 1 = Enable finger present detection
XY_Scale	Sets scaling factor for XY 0 = Disable scaling 1 = Enable scaling

Each feature will be described in detail within the next sections.

3.0 Speed Switching

Speed Switching (SS) allows the sensor to automatically switch the sensor resolution in count per inch (CPI) according to the finger tracking speed. It can be set to either the manual or auto mode. To enter manual mode, the OFN_Engine register 0x60 bit Engine or 7 must be enabled while bit Speed or 6 is set to 0. To enter automatic mode, register OFN_Engine address 0x60, bit Engine or 7 and bit Speed or 6 are set to 1.

3.1 High and Low CPI

The available resolutions (in count per inch, CPI) are 500 cpi, 750 cpi and 1000 cpi. Bit 2:0 in OFN_Resolution register (0x62) set the resolution. Resolution can be further reduced to 250 cpi or increased to 1250 cpi respectively by setting bit 1 and 0 to high in OFN_Speed_Control register (0x63).

3.2 Manual Resolution Setting

There is an option to manually change the resolution of the sensor. To enter this mode, the Speed bit in address 0x60 must be disabled. In manual speed switching mode the external microcontroller (MCU) monitors the reported motion data and changes the sensor resolution accordingly. When the user's finger is navigating slowly, it usually indicates precision cursor placement is desirable so the MCU should set the OFN resolution to low. When the user's finger is navigating at higher speed the OFN resolution should be set to high to accommodate high-speed cursor movements.

Please refer to the register descriptions in the data sheet for settings.

3.3 Automatic Speed Switching

This mode is entered after setting 0x60 bit 7 and 6 to 1. In automatic speed switching mode the sensor will monitor the reported motion and change the resolution automatically. There are also options to change the monitoring rate, wakeup resolution and step-by-step resolution switching control.

3.3.1 Speed Switching Checking Interval

Speed and Resolution checking and switching interval can be incremented from a minimum of 4 ms to a maximum of 16ms in 4 ms step with bits 3 and 2 of register 0x63. The default value is 8 ms; meaning at every 8 ms interval the sensor will check the navigation speed and switch to the applicable resolution described in section 3.3.3.

3.3.2 Wakeup Resolution Control

By writing to register 0x62 bit 5:3, the user can select the sensor wakeup resolution from rest modes. Please be sure to refer to the data sheet register description section for bit values.

3.3.3 Resolution Switching Threshold

Register 0x60 allows user control of the resolution switching according to navigation speed of the finger. There are eight resolution switching threshold registers to facilitate flexible control of the resolution switching. These switching thresholds come in a pair. The registers are named OFN_Speed_STxx from address 0x64 to 0x6B. The register values are derived from the following threshold formula:

Register Threshold (in decimal) = Speed x 8; where speed is in inch/second

As an example of resolution switch control suitable for VGA screen size:

Table 2. Speed Switching Threshold

Switch from	At Speed of	Register	Register Threshold (decimal)	Write value (hex)
250cpi to 500 cpi	> 1 ips	OFN_Speed_ST12 (0x64)	1 x 8 = 8	0x08
500cpi to 250 cpi	< 0.75 ips	OFN_Speed_ST21 (0x65)	0.75 x 8 = 6	0x06
500cpi to 750 cpi	> 8 ips	OFN_Speed_ST23 (0x66)	8 x 8 = 64	0x40
750cpi to 500 cpi	< 1 ips	OFN_Speed_ST32 (0x67)	1 x 8 = 8	0x08
750cpi to 1000 cpi	> 9 ips	OFN_Speed_ST34 (0x68)	9 x 8 = 72	0x48
1000cpi to 750 cpi	< 1.25 ips	OFN_Speed_ST43 (0x69)	1.25 x 8 = 10	0x0a
1000cpi to 1250 cpi	> 10 ips	OFN_Speed_ST45 (0x6A)	10 x 8 = 80	0x50
1250cpi to 1000 cpi	< 9 ips	OFN_Speed_ST54 (0x6B)	9 x 8 = 72	0x48

4.0 Assert or De-assert (A/D)

The Assert/De-assert (A/D) function, when enabled, continuously monitors the shutter values and clears motion data if deemed invalid. The Assert or De-assert functions such that when the finger is off the sensor array, less light from the IR LED is reflected back to the sensor and hence, the sensor shutter will open to a large number. This is the De-assert state. In contrast, when the finger is on the sensor array, more IR LED light will be reflected back to the sensor and the shutter will close down to a small value. This is the Assert state. To activate the A/D function, register OFN_Engine (0x60) bit Assert/Deassert or 5 must be enabled. See Section 2.0

4.1 A/D Control

There are five registers associated with the A/D function. The first register is OFN_AD_CTRL at 0x6D. Bits 2:0 (ST_HIGH 2:0) define the resolution threshold for the A/D High and Low to activate. The default value is 0x04 or 1000 cpi and corresponding speed is 10 ips according to the S/S set earlier in the Resolution Switch Control section 3.3.3. Any speed and resolution detected by the sensor that is higher than 10 ips or 1000 cpi the A/D High thresholds in OFN_AD_ATH_HIGH and OFN_AD_DTH_HIGH are used. When speed is below 10 ips (under resolution 1000 cpi), then A/D Low thresholds in OFN_AD_ATH_LOW and OFN_AD_DTH_LOW are used. This register can be varied according to user preference.

4.2 A/D Threshold

There are 4 registers to change A/D thresholds, 2 registers each for high and low speeds. The A/D high speed registers are OFN_AD_ATH_HIGH and OFN_AD_DTH_HIGH at address 0x6E and 0x6F. The A/D low speed registers are OFN_AD_ATH_LOW and OFN_AD_DTH_LOW at address 0x70 and 0x71.

The threshold value for each of this register is referenced to the sensor shutter value read out at the shutter registers, Shutter_Upper (0x06) and Shutter_Lower (0x07).

The A/D threshold application is further explained in Figure 1. Whenever shutter values are below Assert High threshold of 416, the sensor will report motion. As long as shutter values do not exceed De-assert High threshold of 480, sensor will continue to report motion. When shutter value exceed 480, the De-assert threshold will activate and the sensor ceases to report motion. Only when shutter values drop below Assert High will the sensor resume reporting motion again.

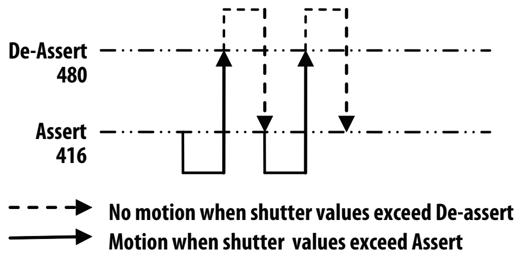

De-Assert 480
Assert 416
No motion when shutter values exceed De-assert
Motion when shutter values exceed Assert

Figure 1. Assert or De-assert Working at High Speed

Notice that if the customized cover over OFN sensor is adopted, which is different to standard OFN module cover material (Lexan 121 21051), due to the variation of surface reflection and transmissivity rate, the re-calculation of Assert/De-assert threshold is strongly recommended for the best tracking performance. (Default register value will be not applicable any more)

Calculation of Assert/De-assert threshold is strongly recommended for the best tracking performance. (Default register value will be not applicable any more)

The Assert/De-assert threshold may be obtained by following the method described in the following. This allows for variations in sensors and external lighting conditions. The OFN sensor setup is tested in a dark room to obtain best threshold values. This eliminates external light sources and ambient light which may distort the shutter values reading. To obtain Assert/De-assert threshold, firstly obtain shutter values for many OFN sensors in a dark room. A finger must be placed on top of the sensor. Secondly the De-assert threshold may be obtained by adding a value of 350 to the shutter value. Thirdly the Assert threshold may be obtained by adding a value of 250 to the shutter value. Then obtain the shutter value when there is no finger on the sensor in the dark room. The value should be higher that the De-assert threshold. If not, then a new value must be obtained. Lastly the tracking performance can be compared between with and without Assert/De-assert. When finger is on the sensor, the tracking performance of the cursor should be smooth. There should not be tracking during both lifted (finger off sensor) and dropped (finger on sensor) finger. An example is shown below:-

Table 3. A/D Threshold Example

Condition	Shutter value
Shutter value in dark, finger ON(average value of 10 sensors)	A3 hex or 163 decimal
Shutter value in dark, finger OFF(average value of 10 sensors)	898 hex or 2200 decimal

Shutter value set	Register address	Formula (decimal)	Write value (hex)
163+ 350 = 513	OFN_AD_DTH_LOW address 0x71	513 / 8 ≈ 60	0x40
163 + 250 = 413	OFN_AD_ATH_LOW address 0x70	413 / 8 ≈ 52	0x34

To simplify the threshold setting process, the same setting value of A/D low speed registers (0x70 and 0x71) can be also used for A/D high speed registers (0x6E and 0x6F), as the high speed navigation can be rarely reached in the real life.

5.0 Finger Presence Detection

The Finger Presence Detection (FPD) detects if a navigation surface is on or off the sensor. Any object or surface covering the sensor array will trigger a "finger" detection as long as it meets the threshold. FPD makes the detection status available at the GPIO pin as well as updates the status bit in the MOTION register. The host can then either report the GPIO status or illuminate an external LED for a visual impact to the user. To enter FPD mode, register OFN_Engine (0x60) bit Finger or 2 must be enabled. See Section 2.0.

FPD is activated in Run mode only and is automatically deactivated in Rest2 and Rest3 modes to conserve power.

5.1 FPD Control

Register OFN_FPD_CTRL (0x75) has 8 bits for varying the FPD control. Bit 6 FPD_POL selects the state of the GPIO pin.

Table 4. FPD_POL states

Bit 6	Description	Finger	GPIO state
0	Active low output	On	0
0	Active low output	Off	1
1	Active high output	On	1
1	Active high output	Off	0

Bits 5 to 0 allow the user to set the FPD state activation threshold. It is based on readings from the shutter values. See Table 5 for examples when selecting shutter values.

Table 5. FPD_POL threshold

FPD Threshold at shutter value	Formula	Decimal	Hex	Bit 5 to 0
640	640 / 32 = 20	20	14	10100
512	512 / 32 = 16	16	10	10000
320	320 / 32 = 10	10	0A	01010

Ambient light with high IR content such as sun light or incandescent lamp shining directly onto the sensor will shorten shutter values. FPD will misinterpret it as a "finger present" condition. Section 9.5 presents additional Firmware FPD solution to improve the finger present recognition under these lighting conditions.

6.0 XY Quantization

Since the OFN sensors reports motion in all directions the XY Quantization feature is useful in applications wanting scroll or rocker mode emulation. In scroll or rocker mode the cursor movements are only in two axis; either up-down or left-right. Section 9.2 and 9.3 describe details for scroll and rocker mode implementations. To enter XYQ mode, register OFN_Engine (0x60) bit XYQ or 4 must be enabled. See Section 2.0.

The XY Quantization (XYQ) feature examines the motion data as a result of the user's finger navigation; deduces the larger motion between the X and Y direction and reports it as a single, on-axis motion. The quantization can be enabled for X only, Y only or both at the same time. When both X and Y quantizations are enabled the data will be reported in either direction based on the larger magnitude detected, with higher priority given to Y axis. To turn on XYQ, set both the OFN_EN and XYQ_EN bits in the OFN_Engine (0x60). Then the XYQ control is provided in OFN_Quantize_Ctrl (0x73); bit 3 for XQ_On and bit 7 for YQ_On. The three bits each of XQ_DIV (2:0) and YQ_DIV (6:4) provide the effective quantization factors of 2^{XQ_DIV} and 2^{YQ_DIV} for both X and Y directions.

However, when motion is close to 45° there is a tendency for the report to oscillate between X and Y. The diagrams below describe the implementation in the OFN to minimize this oscillation. The method defines the regions for X quantization, Y quantization or both where both deltaX and deltaY would be clipped to 0. The boundaries are defined based on a linear function, Y=MX+C where M is the gradient and C is the Y axis intercept. Y quantization will be reported if the absolute Y value is **greater** than or equal to MX+C. X quantization will be reported if the absolute Y value is **less** than M^{-1}X-C. There will be no reporting if motion falls in the area in between. These variables are configurable through register OFN_XYQ_Thres (0x74). Gradient M is set by bit 2 and Y axis intercept C is set with bits 1 and 0. Figure 2 and 3 illustrate the concept.

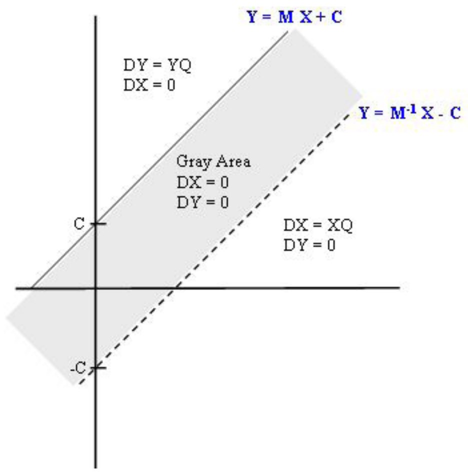


Figure 2. Quantization region where M=1

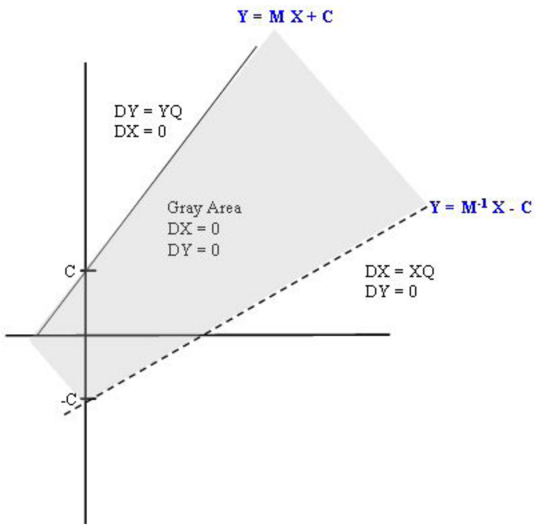
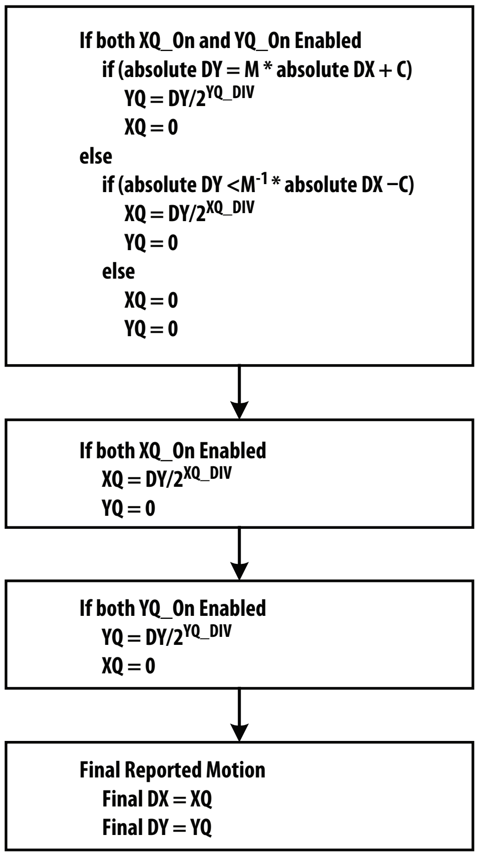


Figure 3. Quantization region where M=2



```
If both XQ_On and YQ_On Enabled
    if (absolute DY = M * absolute DX + C)
        YQ = DY/2^YQ_DIV
        XQ = 0
else
    if (absolute DY <M^-1 * absolute DX –C)
        XQ = DY/2^XQ_DIV
        YQ = 0
    else
        XQ = 0
        YQ = 0

If both XQ_On Enabled
    XQ = DY/2^XQ_DIV
    YQ = 0

If both YQ_On Enabled
    YQ = DY/2^YQ_DIV
    XQ = 0

Final Reported Motion
    Final DX = XQ
    Final DY = YQ
```

Figure 4. Example of XY Quantization

7.0 XY Scaling

XY Scaling sets the sensor to report double the count of either both or single DeltaX and DeltaY values. This can be set in OFN_Speed_Control (0x63) bit 6 for X_scale and bit 7 for Y_scale.

8.0 Sensor Orientation

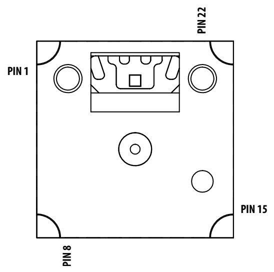
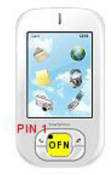
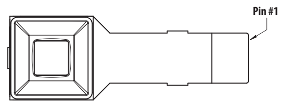
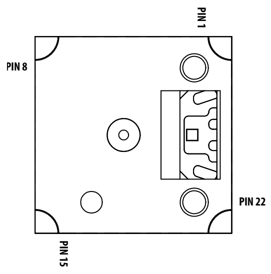
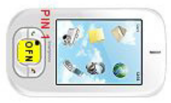
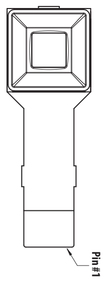
The sensor orientation is configurable via two ways either as *Orient input pin* or as the *OFN_Orientation_Ctrl register (0x77)*. Both controls are always additive to each other.

8.1 Orient Input Pin

"Orient Input Pin" can compensate for a single-direction 90-degree turn of the OFN device. It is used typically to accommodate the Portrait and Landscape Modes or can be used to compensate for a 90-degree clockwise rotation from its standard mounting position. Alternative orientations will need to get compensated by using the Orientation_CTRL register.

In the illustrations below, the Orient pin is set to 1 for the Portrait mode and 0 for the Landscape mode, the sensor will report the followings; see Table 6.

Table 6. Orient input pin state

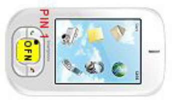
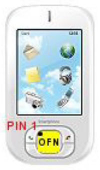
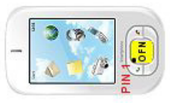
Orient	Part	Package	Mounting Orientation	Finger Direction	Sensor reported
1	ADBS		 (Portrait mode)	Right Left Up Down	X positive X negative Y positive Y negative
	ADBM				
0	ADBS		 (Landscape mode)	Right Left Up Down	X positive X negative Y positive Y negative
	ADBM				

The orientation pin logic 1 has no effect on the data, while orientation logic 0 causes counterclockwise 90-degree rotation of the output data. For a device that can operate in either Portrait or Landscape mode the motion data reporting can be corrected by dynamically changing the logic level on the Orient pin. In addition the OFN_Orientation CTRL register (0x77) can also be used to accomplish the correction.

Orient Pin with and OFN_Orientation_CTRL Register Settings

Portrait mode is the most used handset display mode. In the discussion below it is used as the default for illustration purposes. However, landscape mode can be used as well. Based on Orient pin setup, the value of the OFN_Orientation_CTRL register (0x77) can be adjusted to compensate for the 4 possible mounting orientations of the OFN device relative to the default phone orientation.

Table 7. Two Cases of Rotation

Case 1	Basic Orientation	Case 2
		
Landscape Mode (after Clockwise rotation, Orient pin 1 →0)	Portrait Mode	Landscape Mode (after Counter Clockwise rotation, Orient pin 0 →1)

Case 1. Portrait to Landscape Mode by Clock-Wise Rotation

Table 8. Portrait to Landscape Mode by Clock-Wise Rotation

OFN Orientation	XY_SWAP	Y_INV	X_INV	Portrait Mode	Landscape Mode after rotation
Pin 1 Up left	0	0	0	Orient = 1	Orient = 0
Pin 1 Up right	1	1	0	Orient = 1	Orient = 0
Pin 1 down right	0	1	1	Orient = 1	Orient = 0
Pin 1 down left	1	0	1	Orient = 1	Orient = 0

Case 2. Portrait to Landscape Mode by Counter-Clock-Wise Rotation

Table 9. Portrait to Landscape Mode by Counter-Clock-Wise Rotation

OFN Orientation	XY_SWAP	Y_INV	X_INV	Portrait Mode	Landscape Mode after rotation
Pin 1 Up left	1	0	1	Orient = 0	Orient = 1
Pin 1 Up right	0	0	0	Orient = 0	Orient = 1
Pin 1 down right	1	1	0	Orient = 0	Orient = 1
Pin 1 down left	0	1	1	Orient = 0	Orient = 1

8.2 Orientation Control Register

In addition to the Orient Input pin control, the OFN_Orientation_Ctrl register 0x77 provides additional control for reporting X and Y motion data. Bit 7 XY_Swap allows the sensor to swap the X and Y values. Bit 5 X_INV changes the direction of X values only. Bit 6 Y_INV changes the direction of Y values only. These three bits of the orientation control register will be able to correct for any relative sensor orientation in the phone.

The final reported data by the sensor is after consideration of the Orient pin state. Also to note, if the XY_Swap bit is set together with Y_INV and X_INV, the XY_Swap function has higher precedence. An example is shown below.

If the orient pin is fixed, the orientation control register can be configured for different OFN orientations. The following tables show the combinations of control register settings in the cases of orient pin = 1 and orient pin = 0.

Table 10. Orientation Control Register

Orient Pin = 1

OFN Orientation	XY_SWAP	Y_INV	X_INV
Pin 1 Up left	1	0	1
Pin 1 Up right	0	0	0
Pin 1 down right	1	1	0
Pin 1 down left	0	1	1

Orient Pin = 0

OFN Orientation	XY_SWAP	Y_INV	X_INV
Pin 1 Up left	0	0	0
Pin 1 Up right	1	1	0
Pin 1 down right	0	1	1
Pin 1 down left	1	0	1

9.0 Reference Firmware Description

This section will present the main routines and algorithm which are implemented in the OFN handheld demo unit. It will guide the user to design their own firmware in the mobile devices by employing the similar ideas.

9.1 Power up sequence

The purpose of this routine is to set several unique features of the ADBM or ADBS-A320 sensor for optical finger navigation. The procedure is listed below:

1. Apply power. For single voltage supply (internal regulator enabled), apply VDDA and VDDIO. For dual voltage supply (internal regulator disabled), apply VDDA first. Wait until VDDA has reached stable voltage supply then apply DVDD.
2. Set NCS, Shutdown, and Orient to low. Set IO_Select to low for TWI or high for SPI.
3. If in TWI mode, set A0 and A1 according to Table 1 TWI slave address in the datasheet. This step is skipped if SPI mode is used.
4. Drive NRST low then high. TWI slave address will only be selected after a NRST toggle is applied when A0 and A1 are set.
5. Perform soft reset by writing 0x5A to address 0x3A.
6. Write 0xE4 to address 0x60.
7. Set Speed Switching, write 0x62 with 0x12, 0x63 with 0x0E, 0x64 with 0x08, 0x65 with 0x06, 0x66 with 0x40, 0x67 with 0x08, 0x68 with 0x48, 0x69 with 0x0A, 0x6A with 0x50, 0x6B with 0x48.
8. Set Assert/De-assert, write 0x6D with 0xC4, 0x6E with 0x34, 0x6F with 0x3C, 0x70 with 0x18, 0x71 with 0x20.
9. Set Finger Presence Detection threshold, write 0x75 with 0x50.
10. IF XY Quantization is used, then write 0x73 with 0x99 and 0x74 with 0x02.
11. Write 0x10 to register 0x1C. This will activate Burst mode. If Burst mode not used then skip this step.
12. Read from registers 0x02, 0x03 and 0x04 (or read these same 3 bytes with Burst mode) one time regardless the state of the motion pin.
13. Write 0x1A with 0x00 (default value) to set the LED drive current 13mA.

Please refer to the reference firmware codes in the function of *Power_Up_Sequence() (File: ADBM_A320.c)* for the exact implementation details.

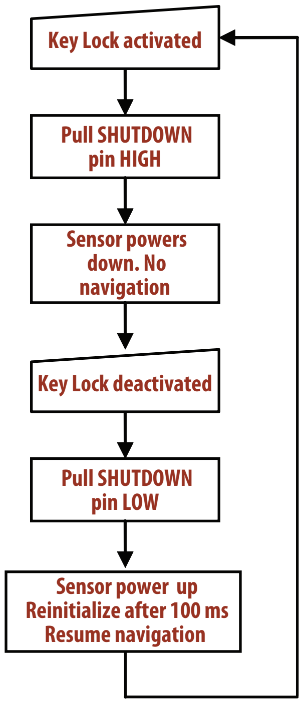
9.2 Power Saving Consideration

Handheld devices are typically battery powered and as such most users demand long battery life This section recommends the power up and power down procedure for the Avago OFN sensor to help the device manufacturers to conserve battery power while integrating the OFN sensors into their devices.

The A320 sensor can be shutdown with a HIGH issued to the SHTDWN pin. SHTDWN should be controlled by the system controller when navigation is no longer required. The OFN sensor consumes only around 3 uA of current during the shutdown state in dual supply mode. During the shutdown state, the DVDD, VDDIO should be maintained at a minimum of 1.65 V and VDDA should be maintained at a minimum of 2.6 V.

In order to reinitialize the sensor from the shutdown state, the SHTDWN pin needs to be pulled LOW. It takes at least 100 ms to completely reinitialize the sensor. Reinitializing the sensor from the shutdown state via the SHTDWN pin will retain all register data that were written to the sensor prior to shutdown. However, resetting the sensor either via Soft_RESET register or through the NRST pin will reset all registers data back to their default values.

Figure 5 illustrates a basic power-up and power-down procedure:


Key Lock activated
Pull SHUTDOWN pin HIGH
Sensor powers down. No navigation
Key Lock deactivated
Pull SHUTDOWN pin LOW
Sensor power up Reinitialize after 100 ms Resume navigation

Notes:
1. The Soft_RESET register (0x3a) should not be written to throughout the shutdown and re-initialization process. Resetting the sensor clears all data values stored in the register.
2. The NRST pin (pin 10) should not be pulled LOW throughout the shutdown and re-initialization process.
3. Wake up time is at least 100 ms.

Figure 5. Basic power-up and power-down procedure

Shutdown Mode

OFN sensor provides a power saving mode called *shutdown mode*. The following table shows the power current in various operational modes and it indicates up to 90% power saving in Shutdown mode compared to Run 2 mode under Internal regulator disabled case.

		Internal regulator Disabled		
Parameter		**Typical**	**Max**	**Units**
DC average supply current in Run mode	Total	3.10	4.30	mA
DC average supply current in Rest1 mode	Total	0.40	0.70	mA
DC average supply current in Rest2 mode	Total	0.10	0.20	mA
DC average supply current in Rest3 mode	Total	0.04	0.15	mA
Analog Shutdown Supply Current	$I_{DDSHTDWN}$ V_{DDA}	0	2	µA
	$I_{DDSHTDWN\ LED+}$	0	0.1	µA
Digital Shutdown Supply Current	$I_{DDSHTDWN}$ DV_{DD}	3	15	µA

Hence the use of shutdown mode is strongly recommended for the power saving purpose.

9.3 Reference Firmware Routines

The Reference Firmware is developed in C with a Silicon Laboratory (SiLab) 8051F347 as target. The 8051F437 is chosen for its generic nature. All the critical routines related to navigation are in two C files: rocker.c and sequence.c. Other files, such as timer.c and i2c.c, etc. are responsible for the system initialization, setup, and communication.

In addition an USB module is adapted to demonstrate the functionalities and operating modes of the handheld demo unit. It is designed and built with a JTAG connector so firmware can be updated.

The C project is compiled in a Keil environment. After the code is compiled, linked and downloaded into the 8051F347 the handheld demo can be connected it to a PC via USB. No special driver is needed as the device will be enumerated as a HID class device by Windows. Only the critical functions are introduced in this section.

Figure 6 gives the overview of data flow between the OFN sensors and PC through the communication modules SPI (TWI) and USB:

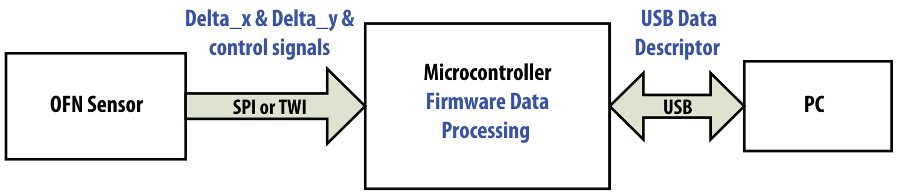


Figure 6. OFN Handheld Demo Architecture

Figure 7 gives a firmware overview showing the key routines.

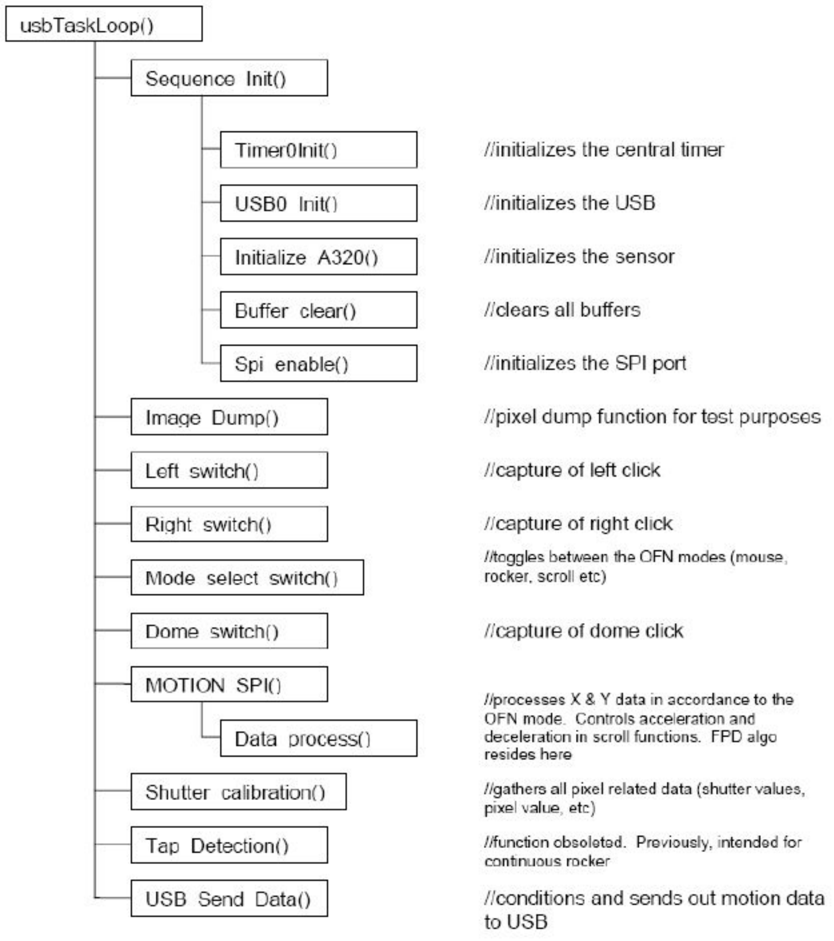

usbTaskLoop()
Sequence Init()
Timer0Init()
//initializes the central timer
USB0 Init()
//initializes the USB
Initialize A320()
//initializes the sensor
Buffer clear()
//clears all buffers
Spi enable()
//initializes the SPI port
Image Dump()
//pixel dump function for test purposes
Left switch()
//capture of left click
Right switch()
//capture of right click
Mode select switch()
//toggles between the OFN modes (mouse, rocker, scroll etc)
Dome switch()
//capture of dome click
MOTION SPI()
Data process()
//processes X & Y data in accordance to the OFN mode. Controls acceleration and deceleration in scroll functions. FPD algo resides here
Shutter calibration()
//gathers all pixel related data (shutter values, pixel value, etc)
Tap Detection()
//function obsoleted. Previously, intended for continuous rocker
USB Send Data()
//conditions and sends out motion data to USB

Figure 7. Firmware structure and routines

1. **Initialization Sequence**

   This function and its sub-functions such as Timer0Init ( ), USB0_Init ( ), Initialize_A320 ( ), Buffer_clear(), spi_enable ( ) initialize the system and reset some variables.

   Please refer to the codes in the function of *Sequence_Init() (File: sequence.c)* for the exact implementation details.

2. **Right Button Click**

   This function deals with the communication mode switching and right-click operation on demo kit. By pressing the right button for a few seconds, the user can switch between the TWI and SPI communication mode.

   The user can assign its own keyboard shortcut for the right-click operation.

   Please refer to the reference firmware codes in the function of *Right_Switch () (File: sequence.c)* for the exact implementation details.

3. **Left Button Click**

   The user can assign its own keyboard shortcut for left-click operation on demo kit.

   Please refer to the codes in the function of *Left_Switch () (File: sequence.c)* for the exact implementation details.

4. **Motion Detection based on SPI and TWI**

   This is the main firmware data collecting and processing part combined with the acceleration algorithm for various user operating modes such as single step and multiple step rocker modes and the software FPD algorithm.

   Section 9.4 will provide more detailed information about how the different operation modes are implemented.

   Please refer to the reference firmware codes in the function of *Motion_SPI () and Motion_TWI() (File: sequence.c)* for the exact implementation details.

5. **Data Processing and Command Output**

   The delta X and delta Y information will be first processed here according to the various operation modes.

   Since each operation mode has its own user features, hence the implementation of each feature is different from the others. For instance, the single step will only consider the single delta X and delta Y information, but in the multiple step mode, the accumulated delta X and delta Y should be considered to remember the whole length of finger swipe on OFN sensor.

   Please refer to the codes in the function of *data_process() (File: sequence.c)* for the exact implementation details.

6. **USB_Send_Data ( )**

   This function formulates the conditions and determines whether to send the motion data in mouse or keyboard format. Examples of Mouse data and Keyboard data are shown below:

Table 11. Samples of different USB sent data

Example: Mouse data

```
IN_PACKET [0]     = MOUSE_MODE;
IN_PACKET [1]     = buttons;
IN_PACKET [2]     = count_x;
IN_PACKET [3]     = -count_y;
IN_PACKET [4]     = count_z;
IN_BUFFER.Ptr     = IN_PACKET;
IN_BUFFER.Length  = MOUSE_DATA_LENGTH;
```

Example: Keyboard data

```
IN_PACKET [0]     = KEY_MODE;
IN_PACKET [1]     = 0;
IN_PACKET [2]     = 0;
IN_PACKET [3]     = keyboard_dir;
IN_PACKET [4]     = 0;
IN_PACKET [5]     = 0;
IN_PACKET [6]     = 0;
IN_PACKET [7]     = 0;
IN_PACKET [8]     = 0;
IN_BUFFER.Ptr     = IN_PACKET;
IN_BUFFER.Length  = KEY_DATA_LENGTH;
```

Please refer to the reference firmware codes in the function of *USB_Send_Data() (File: sequence.c)* for the exact implementation details.

9.4 Operating Modes

Based on the specific application requirement the different OFN operating modes can be dynamically activated in real time to best suit the need of the user. For instance, a mobile phone user may switch from navigating an on-screen numeric keypad using the mouse mode to selecting a song from the MP3 play list using the scroll mode. And as he scrolls the list if he hold his finger over the OFN for a few seconds the list can be set to move up or down at an acceleration rate.

The following operating modes are implemented in the latest OFN A320 Reference Firmware.

1. Mouse Mode
2. Single Step Rocker (SSR) Mode
3. Multiple Step Rocker / Inertia Rocker (MSR/IR) Mode
4. Scroll and Free Scroll Mode
5. Joystick Mode
6. 8 Way Rocker Mode

Please refer to the codes in the function of d*ata_process() (File: sequence.c)* for the exact implementation details.

Mouse Mode

In mouse mode, the cursor on the screen (either on PC monitor or mobile phone screen) moves like a regular mouse.

The register values of delta_x and delta_y are read and then sent to the USB communication routine in which the formatted data is sent to the host.

Notice that the sensitivity of the cursor movement in mouse mode can be adjusted by multiplying delta_x and delta_y with certain integer value. The bigger the integer value is, the more distance (more agile) the cursor moves

Single or Multiple Step Rocker Mode

Single Step Rocker mode produces one motion value when finger swipes across the sensor once and lifted.

Multiple Step Rocker mode results in more than one motion value depends on how long the finger swipes across the sensor. This is useful if the intention is emulate multiple rocker steps.

The only difference between Single and Multiple Step Rocker modes is the toggle processing function. Function toggle_step2() for Multiple Step Rocker mode periodically checks timer0 (polling_period = 50ms) to decide if the additional step action is needed. However function toggle_step() will only execute once for each swipe.

Figure 8 illustrates the relationship between the main data processing function with 8 ms polling period and toggle_step() and toggle_step2(). These functions work together to process accumulated delta_x and delta_y motion data Rocker mode.

Notice that in the toggle_step() and toggle_step2() functions, the *threshold* is set to decide whether the accumulated data account for a step action. The threshold therefore is derived from a set of navigational preferences such as sensitivity and precision. Generally speaking, by spreading out the duration at which the motion data is compared with the threshold, the sensitivity is reduced and the rocker behavior would be more precise. Optimal performance is achieved by fine-tuning the various polling period and threshold.

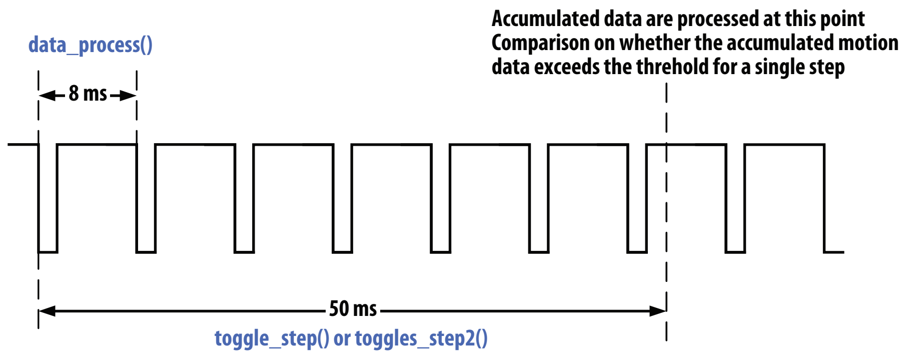

data_process()
8 ms
Accumulated data are processed at this point
Comparison on whether the accumulated motion data exceeds the threhold for a single step
50 ms
toggle_step() or toggles_step2()

Figure 8. Polling theory of data_process() and toglle_step() functions.

Figure 9 shows the flow chart of the implementation of Single and Multiple Step Rocker modes.

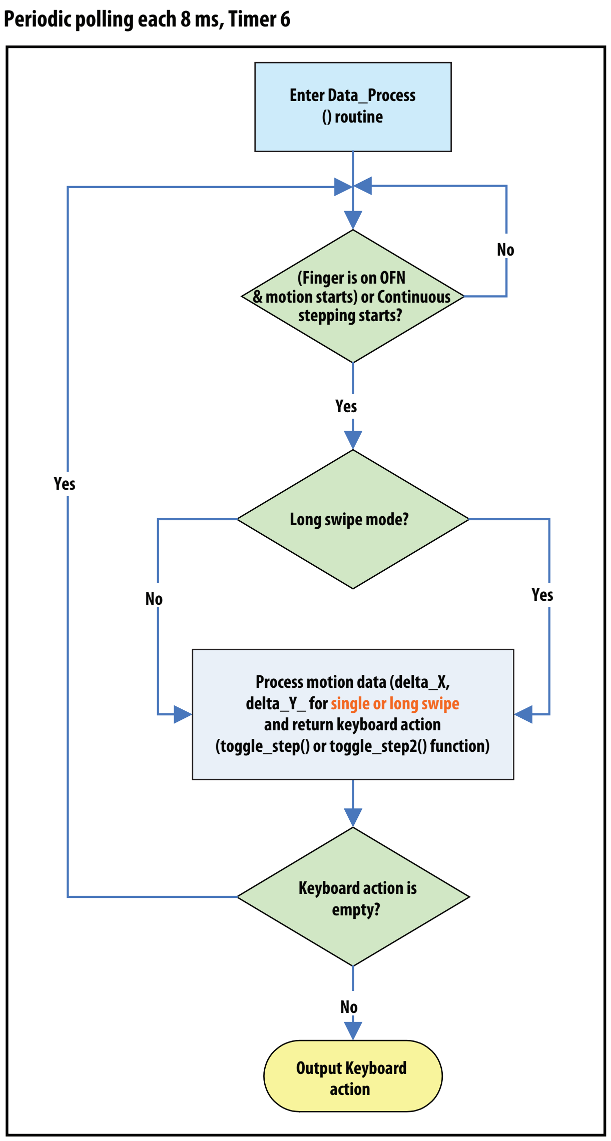

Periodic polling each 8 ms, Timer 6
Enter Data_Process () routine
(Finger is on OFN & motion starts) or Continuous stepping starts?
No
Yes
Long swipe mode?
No
Yes
Yes
Process motion data (delta_X, delta_Y_ for single or long swipe and return keyboard action (toggle_step() or toggle_step2() function)
Keyboard action is empty?
No
Output Keyboard action

Figure 9. Implementation of single and multiple step rocker modes

Inertia Swipe

Inertia swipe is a special function based on the multiple step rocker setting. When the user swipes the sensor, and removes the finger from the surface at once, this action is recognized by the firmware as triggering the Inertia Swipe. The firmware will produce certain amount of rocker movements depending on the direction and the distance of the swipe on the sensor surface.

The implementation of Inertia Swipe is based on two separate trigger events:

1. Swipe distance, and
2. Finger lift from the sensor surface

The swipe distance is calculated from the total accumulated deltaX and deltaY counts (counts/cpi). The finger lift is detected by the FPD firmware algorithm described in section 9.5.

"Hold and Acceleration" Function (also as "Auto Repetition" or "Continuous")

"Hold and Acceleration" activates when the user completes one swipe and then hold the finger on the sensor surface without lifting. The firmware detects this scenario and initiates the step motion with accelerated rate.

The resulting cursor movement is best described in a real-life application. For example, a user is searching for the songs in the play list of a MP3 player. If the user swipes the sensor, the list will move in the direction of the swipe. And if the user holds the finger on the sensor for a period time after the swipe (with no finger lift), the highlighted item will continue to move in the same direction at an accelerated rate.

The following parameters can be adjusted for the repetitious rate.

Table 12. Parameter list for tuning purpose

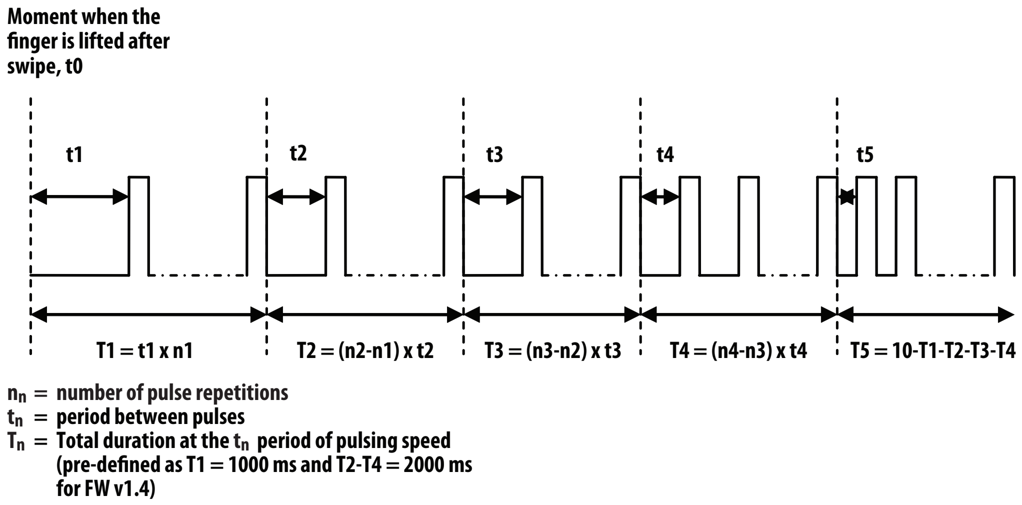
No	Parameter	Function	Defined in	Description
1	t1	periodic time between two pulses	sequence.c	Time between pulses in period T1
2	t2	periodic time between two pulses	sequence.c	Time between pulses in period T2
3	t3	periodic time between two pulses	sequence.c	Time between pulses in period T3
4	t4	periodic time between two pulses	sequence.c	Time between pulses in period T4
5	t5	periodic time between two pulses	sequence.c	Time between pulses in period T5
6	n1	Repetition rate for t1	sequence.c	
7	n2-n1	Repetition rate for t2	sequence.c	
8	n3-n2	Repetition rate for t3	sequence.c	
9	n4-n3	Repetition rate for t4	sequence.c	

Figure 10 visualizes how t1 – t5 and n1 – n4 are defined for repetition rate. Here is a short explanation:

1. t1 – t5 define time between two pulses in the respective period T1 to T5, T1 starts when the finger is held on the OFN (swiped finished).
2. Four different repetition n1 – n4 are used to calculate the current repetition rate.
3. The total hold time can be calculated as the sum of T1 – T5 accordingly.
4. The maximum continuous rocker time is 10 seconds
5. The following comments from code give a visual explanation:

Table 13. Sample definition of repetition rate

```
// START| Speed 1 | Speed 2 | Speed 3 | Speed 4 | Speed 5
// START| n1 x t1 | (n2-n1) x t2 | (n3-n2) x t3 | (n4-n3) x t4 | Polling rate based on t5
//
// Default Setting:
// START| Speed 1 | Speed 2 | Speed 3 | Speed 4 | Speed 5
// START| 2 x 500 | (10-2) x 250 | (26-10)x 125 | (58-26) x 62 | Polling rate of 31ms
// | 1000 | 2000 | 2000 | 1984 | beyond 'Speed 4'
// n=repetition
```



Moment when the finger is lifted after swipe, t0
t1
t2
t3
t4
t5
T1 = t1 x n1
T2 = (n2-n1) x t2
T3 = (n3-n2) x t3
T4 = (n4-n3) x t4
T5 = 10-T1-T2-T3-T4
n_n = number of pulse repetitions
t_n = period between pulses
T_n = Total duration at the t_n period of pulsing speed (pre-defined as T1 = 1000 ms and T2-T4 = 2000 ms for FW v1.4)

Figure 10. Repetition rate setup in continuous rocker mode

Polling Period Adjustment for "Steps per Swipe"

The "polling_period" parameter, which is defined in the file: *Rocker.c* with default value 50ms, can be used as the setup for "steps per swipe".

In addition:

1. "steps per swipe" is the function of "speed of swipe (counts per ms)" and "polling period (ms)".
2. If "speed of swipe" is known and by changing the "polling period", the "steps per swipe" can be adjusted.
3. The step number is calculated as the multiple of polling periods. Counts are accumulated and compared against a fixed threshold for every polling period.
4. If polling period is fixed higher speed of swipe produces more steps.

Scroll and Free Scroll Mode

Scroll and Free Scroll mode can also be emulated using the firmware by detecting changes in motion data.

In free scroll mode, after a long swipe the cursor will keep scrolling until the finger taps the sensor again. Figure 11 shows how it works together with normal scroll mode. If the parameter free_scroll = 1, the timer11 will have a longer time-out, which means, the scroll action will automatically go longer for a certain period of time.

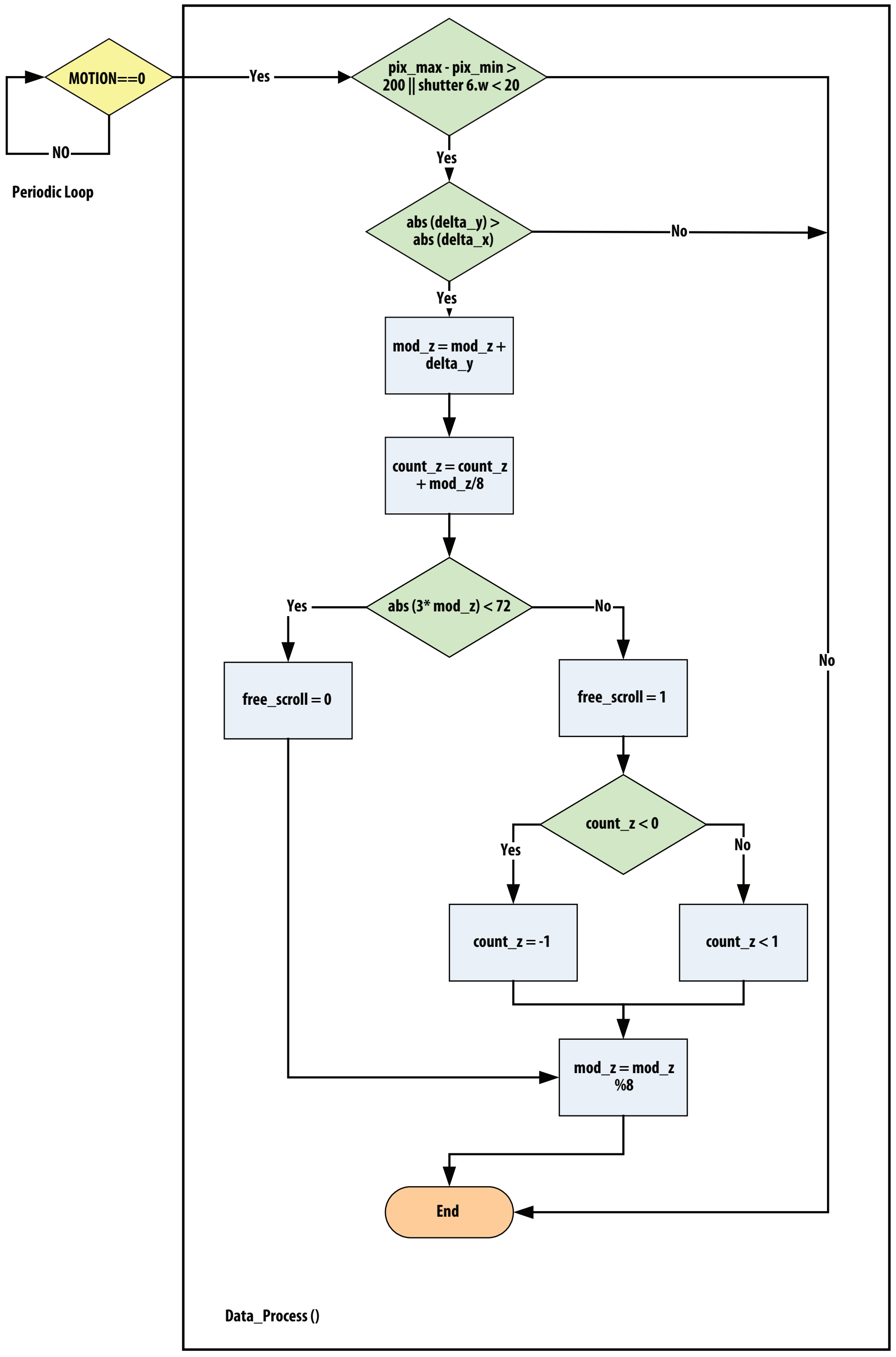


Figure 11. Scroll and Free Scroll Mode

Joystick Mode

Joystick mode is implemented with the Reference Firmware in the demo kit to emulate the joystick operations in a Windows environment.

Since the joystick mode belongs to the class of HID (Human Interface Device) under Windows, so the corresponding device descriptor of HID must be configured for joystick mode, as the following codes illustrates.

Table 14. Additional HID descriptor codes for joystick mode in firmware

```
0x05, 0x01, // Usage Page (Generic Desktop)
0x09, 0x04, // Usage (Joystick)
0xA1, 0x01, // Collection (Application)
0x85, 0x04, // REPORT ID (0x04)

0x09, 0x01, // Usage (Pointer)
0xA1, 0x00, // Collection (Physical)
0x05, 0x01, // Usage Page (Generic Desktop)
0x09, 0x30, // Usage (X)
0x09, 0x31, // Usage (Y)
0x15, 0x00, // Logical Minimum (0)
0x26, 0xFF, 0x00, // Logical Maximum (255)
0x35, 0x00, // Physical Minimum (0)
0x46, 0xFF, 0x00, // Physical Maximum (255)
0x65, 0x00, // Unit (None)
0x75, 0x08, // Report Size (8)
0x95, 0x02, // Report Count (2)
0x81, 0x02, // Input (Data, Variable, Absolute)

0x05, 0x09, // Usage Page (Button)
0x15, 0x00, // Logical Minimum (0)
0x25, 0x01, // Logical Maximum (1)
0x19, 0x01, // Usage Minimum (1)
0x29, 0x08, // Usage Maximum (8)
0x95, 0x08, // Report Count (8)
0x75, 0x01, // Report Size (1)
0x81, 0x02, // Input (Data, Variable, Absolute)

0xC0, // End Collection (Physical)
0xC0, // End Collection (Application)
```

Within the demo kit, the joystick mode is equipped with 8-way direction selection and certain button clicking settings. The 8-way direction selection is implemented by applying the C51 math function *atan2(x,y)*, which returns the arc tangent in radians of *y/x* based on the signs of both values to determine the correct quadrant. Here *x and y are the total displacement of one finger movement on the OFN sensor*. The following list presents the reporting direction corresponding to the calculated radian value:

Table 15. 8-way joystick direction mapping to radian value

```
// 337.5 deg    -> 022.5 deg ==> RIGHT
// 022.5 deg    -> 067.5 deg ==> UP-RIGHT
// 067.5 deg    -> 112.5 deg ==> UP
// 112.5 deg    -> 157.5 deg ==> UP-LEFT
// 157.5 deg    -> 202.5 deg ==> LEFT
// 202.5 deg    -> 247.5 deg ==> DOWN-LEFT
// 247.5 deg    -> 292.5 deg ==> DOWN
// 292.5 deg    -> 337.5 deg ==> DOWN-RIGHT
```

Notice that within joystick mode, the certain windows joystick software should be used to test its configuration. No normal mouse or keyboard operation will be available under this case.

8 Way Rocker Mode

8 way rocker mode provides the possibility of 8 direction's moving: UP, DOWN, RIGHT, LEFT, UP-RIGHT, UP-LEFT, DOWN-LEFT and DOWN-RIGHT. It simulates the 45 degree triangle cursor movement over the menu selection, for instance on a cell phone screen with different icons.

This mode can be implemented as either single rocker or multiple step rocker setting. The main idea is to compose any two basic movements, for sample, UP and DOWN in one polling period, to produce "one time" movement like UP-DOWN. In this case, the thresholds for X and Y coordination are used to differentiate the various basic movements.

9.5 Firmware FPD (Finger Presence Detection)

As explained in section 5, FPD (Finger Presence Detection) is a signal to indicate whether the sensor is covered by a surface (for example, a finger) by using the shutter values. However, ambient light with high IR content such as sun light or incandescent lamp shining directly onto the sensor causes very short shutter values. This fools the sensor to mistakenly conclude the presence of a finger.

The FPD algorithm introduced in firmware attempts to minimize the effect of the ambient light interference. The code implementation expands the criteria for a "finger on" condition to enhance the FPD effect. To summarize, the code performs the followings:

1. Checks if there is a motion occurrence on the sensor at the different operating conditions.
2. Reads and collects statistical data from registers such as shutter, pixel_min, pixel_max, pixel_average and squal value.
3. Based on the statistics collection, determines if ambient light is coming from the side and if the finger has left the navigation area
4. Periodically checks the shutter ratio value at different LED driving currents (implemented in *LED_driving_current_switch (Figure 12)*)
5. Reports FPD and display FPD status (LED indicator on handheld demo unit)

Please refer to Figure 13 for the FPD algorithm flow chart.

Notice that switching the LED driving current in the firmware FPD solution could slightly increase the power consumption (supply current). For example, the current increases by 1mA at VDD_LED or at VDDA assuming VDD_LED source is tied to VDDA.

Avago's recommendation is to use *time-outs* from the end of data reporting to indicate the finger-up event, and the commencement of new data reporting to indicate finger-on.

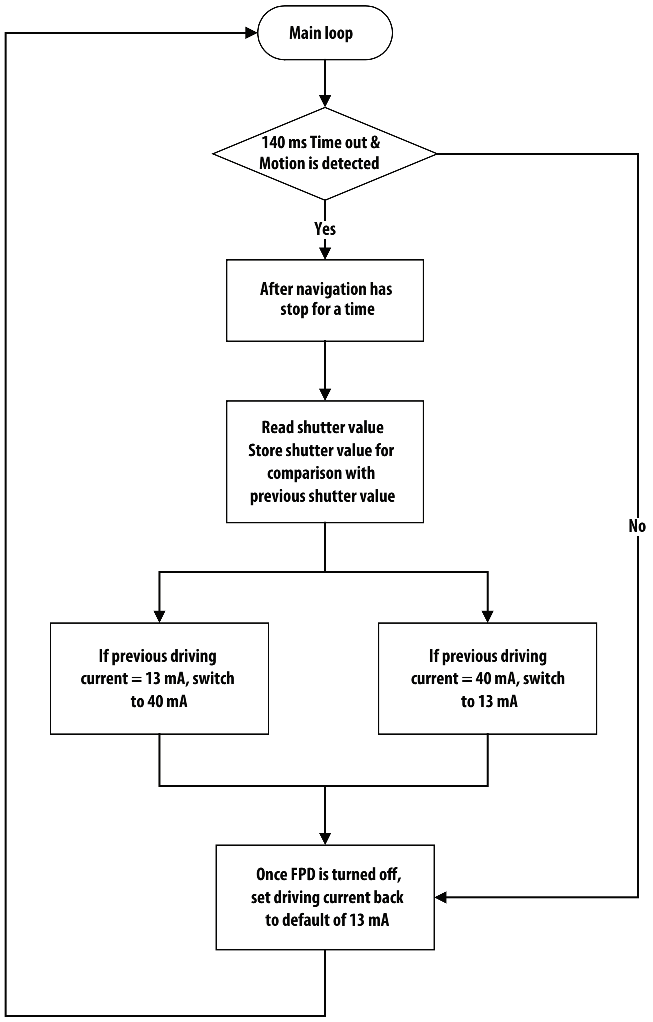
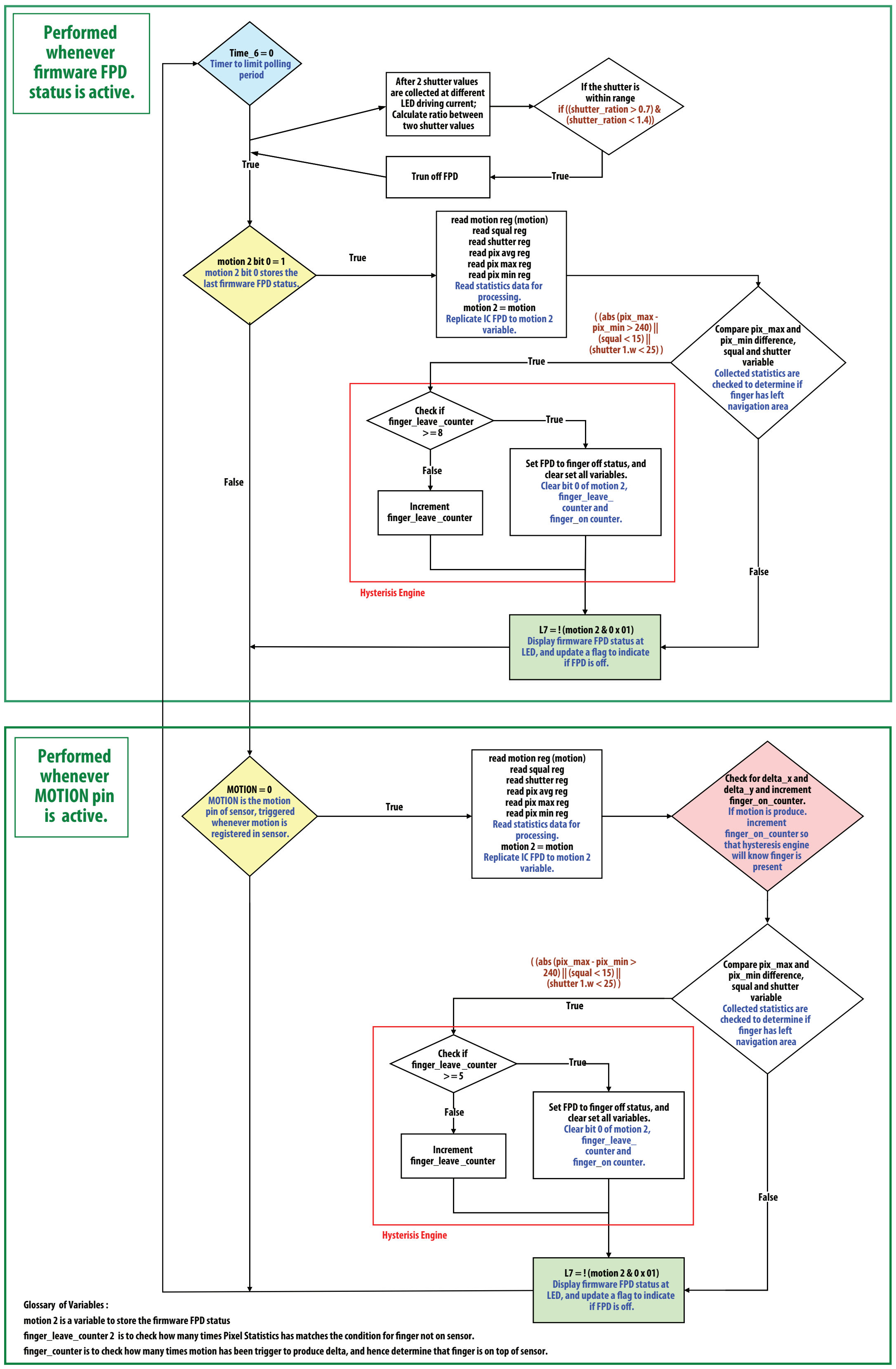

Main loop
140 ms Time out & Motion is detected
Yes
No
After navigation has stop for a time
Read shutter value Store shutter value for comparison with previous shutter value
If previous driving current = 13 mA, switch to 40 mA
If previous driving current = 40 mA, switch to 13 mA
Once FPD is turned off, set driving current back to default of 13 mA

Figure 12. Flow chart of function LED_driving_current_switch()

Please refer to the reference firmware codes in the function of *Motion_SPI() (File: sequence.c)* for the exact implementation details.


Performed whenever firmware FPD status is active.
Time_6 = 0
Timer to limit polling period
After 2 shutter values are collected at different LED driving current; Calculate ratio between two shutter values
If the shutter is within range
if ((shutter_ration > 0.7) & (shutter_ration < 1.4))
True
Trun off FPD
True
motion 2 bit 0 = 1
motion 2 bit 0 stores the last firmware FPD status.
True
read motion reg (motion)
read squal reg
read shutter reg
read pix avg reg
read pix max reg
read pix min reg
Read statistics data for processing.
motion 2 = motion
Replicate IC FPD to motion 2 variable.
((abs (pix_max - pix_min > 240) || (squal < 15) || (shutter 1.w < 25))
Compare pix_max and pix_min difference, squal and shutter variable
Collected statistics are checked to determine if finger has left navigation area
True
Check if finger_leave_counter >=8
True
Set FPD to finger off status, and clear set all variables.
Clear bit 0 of motion 2, finger_leave_ counter and finger_on counter.
False
Increment finger_leave_counter
Hysterisis Engine
False
False
L7 = ! (motion 2 & 0 x 01)
Display firmware FPD status at LED, and update a flag to indicate if FPD is off.
Performed whenever MOTION pin is active.
MOTION = 0
MOTION is the motion pin of sensor, triggered whenever motion is registered in sensor.
True
read motion reg (motion)
read squal reg
read shutter reg
read pix avg reg
read pix max reg
read pix min reg
Read statistics data for processing.
motion 2 = motion
Replicate IC FPD to motion 2 variable.
Check for delta_x and delta_y and increment finger_on_counter.
If motion is produce. increment finger_on_counter so that hysteresis engine will know finger is present
((abs (pix_max - pix_min > 240) || (squal < 15) || (shutter 1.w < 25))
True
Compare pix_max and pix_min difference, squal and shutter variable
Collected statistics are checked to determine if finger has left navigation area
Check if finger_leave_counter >=5
True
Set FPD to finger off status, and clear set all variables.
Clear bit 0 of motion 2, finger_leave_ counter and finger_on counter.
False
Increment finger_leave_counter
Hysterisis Engine
False
L7 = ! (motion 2 & 0 x 01)
Display firmware FPD status at LED, and update a flag to indicate if FPD is off.

Glossary of Variables :
motion 2 is a variable to store the firmware FPD status
finger_leave_counter 2 is to check how many times Pixel Statistics has matches the condition for finger not on sensor.
finger_counter is to check how many times motion has been trigger to produce delta, and hence determine that finger is on top of sensor.

Figure 13. External FPD Algorithm Flow Chart

10.0 Pixel Grab Function

OFN sensor provides the function of *Pixel Grab* for test and diagnostic purpose. It can be also understood as Picture Capture function from the sensor's view. It shows what the sensor can "see" from the OFN surface. One example is shown below:

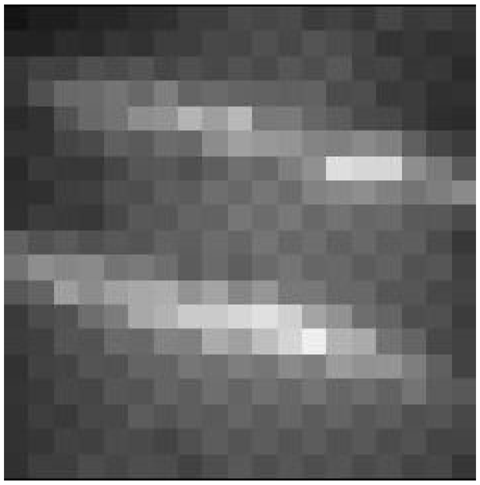


Figure 14. Pixel Grab Picture Example (Finger print fringe shows high light area)

The sensor will read out the contents of the pixel array, one pixel per register frame (Register Pixel_Grab address 0x0b). The 8-bits value presents the gray value of each pixel (total 256 levels). To start a pixel grab, write anything to this register to reset the pointer to pixel 0,0. Then read the PIXRDY bit in the Motion register. When the PIXRDY bit is set, there is valid data in this register to read out. After the data in this register is read, the pointer will automatically increment to the next pixel. Reading may continue indefinitely; once a complete frame's worth of pixels has been read (19 X 19 = 361), PIXFIRST bit will be set to high to indicate the start of the first pixel and the address pointer will start at the beginning location again.

The corresponding flow chart is shown below.

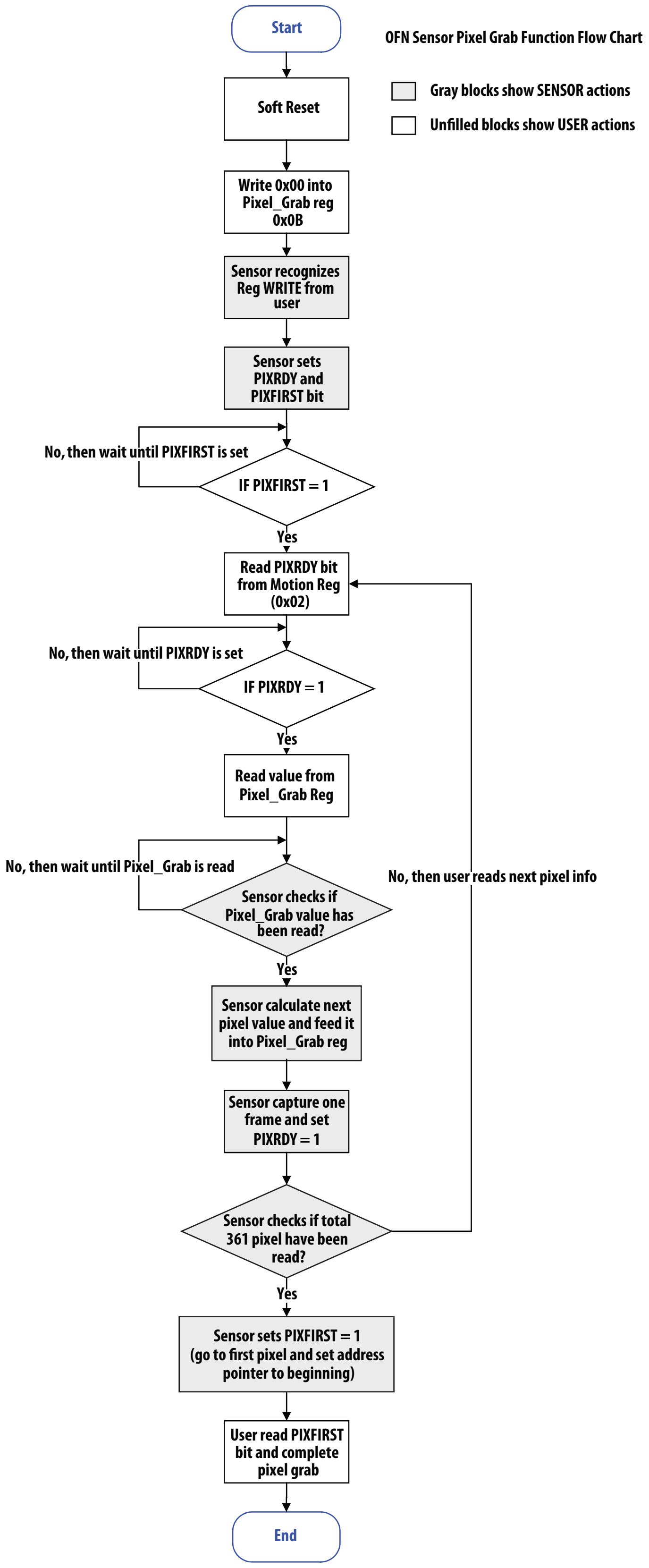

OFN Sensor Pixel Grab Function Flow Chart
Gray blocks show SENSOR actions
Unfilled blocks show USER actions
Start
Soft Reset
Write 0x00 into Pixel_Grab reg 0x0B
Sensor recognizes Reg WRITE from user
Sensor sets PIXRDY and PIXFIRST bit
No, then wait until PIXFIRST is set
IF PIXFIRST = 1
Yes
Read PIXRDY bit from Motion Reg (0x02)
No, then wait until PIXRDY is set
IF PIXRDY = 1
Yes
Read value from Pixel_Grab Reg
No, then wait until Pixel_Grab is read
Sensor checks if Pixel_Grab value has been read?
No, then user reads next pixel info
Yes
Sensor calculate next pixel value and feed it into Pixel_Grab reg
Sensor capture one frame and set PIXRDY = 1
Sensor checks if total 361 pixel have been read?
Yes
Sensor sets PIXFIRST = 1 (go to first pixel and set address pointer to beginning)
User read PIXFIRST bit and complete pixel grab
End

Figure 15. Pixel Grab Operation Flow Chart